AN ENHANCED ANDROID BOTNET DETECTION APPROACH USING FEATURE REFINEMENT

SHAHID ANWAR

DOCTOR OF PHILOSOPHY

(COPUTER SCIENCE)

UNIVERSITI MALAYSIA PAHANG

UNIVERSITI MALAYSIA PAHANG

DECLARATION OF THESIS AND COPYRIGHT

Author's Full Name	:	SHAHID ANWAR
Date of Birth	:	DECEMBER 25, 1984
Title	:	AN ENHANCED ANDROID BOTNET DETECTION APPROACH USING FEATURE REFINEMENT
Academic Session	:	SEMESTER 2, 2018/2019

I declare that this thesis is classified as:

☐ CONFIDENTIAL	(Contains confidential information under the Official Secret Act 1997) *
☐ RESTRICTED	(Contains restricted information as specified by the organization where research was done) *
☑ OPEN ACCESS	I agree that my thesis to be published as online open access (Full Text)

I acknowledge that Universiti Malaysia Pahang reserve the right as follows:

1. The Thesis is the Property of Universiti Malaysia Pahang
2. The Library of Universiti Malaysia Pahang has the right to make copies for the purpose of research only.
3. The Library has the right to make copies of the thesis for academic exchange.

Certified by:

(Student's Signature)	(Supervisor's Signature)
WM1798073	Assoc. Prof. Dr. Mohamad Fadli Zolkipli
New IC/Passport Number	Name of Supervisor
Date:	Date:

NOTE : * If the thesis is CONFIDENTIAL or RESTRICTED, please attach a thesis declaration letter.

SUPERVISOR'S DECLARATION

I hereby declare that I have checked this thesis, and in my opinion, this thesis is adequate in terms of scope and quality for the award of the degree of Doctor of Philosophy (Computer Science).

____

(Supervisor's Signature)

Full Name : Ts Dr MOHAMAD FADLI ZOLKIPLI

Position : Associate Professor, Deputy Dean Academic

Date:

STUDENT'S DECLARATION

I hereby declare that the work in this thesis is based on my original work except for quotations and citation which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at Universiti Malaysia Pahang or any other institutions.

____
(Student's Signature)

Full Name	: SHAHID ANWAR
ID Number	: PCC13006
Date:	

AN ENHANCED ANDROID BOTNET DETECTION APPROACH USING FEATURE REFINEMENT

SHAHID ANWAR

Thesis submitted in fulfilment of the requirements
for the award of the degree of
Doctor of Philosophy
(Computer Science)

Faculty of Computer Systems & Software Engineering
UNIVERSITI MALAYSIA PAHANG

APRIL 2019

DEDICATION

“And We have not sent you, [O Muhammad], except as a mercy to the worlds."

(Quran: 21:107)

DEDICATED TO RAHMAT-UL LIL-ALAMEEN (SAL-LAL-LAHO ALEHI WA ALIHI WASALLAM)

ACKNOWLEDGEMENT

First and foremost, I would like to express my deepest gratitude to my supervisors TS. Dr. Mohamad Fadli Zolkipli and Prof Dr. Jasni Mohamad Zain who have always supported, encouraged, and guided me throughout my Ph.D. journey.

I would like to acknowledge my collaborators for their constant support and invaluable inputs to my research. Special thanks to my sister Dr. Zakira Inayat (UET Peshawar) for supporting, motivating, and collaborating with me throughout these years. She offered a lot of help and advice on my research as well as wonderful friendship. I would like to say thanks to Dr. Nawsher Khan (King Khalid University, Abha, KSA) for supporting me during my initial days in Malaysia. I am grateful to my present and former FSKKP Lab members for their accompany and helpful discussions from time to time, especially Dr. Julius Beneoluchi Odili (Anchor University Lagos, Nigeria), Dr Hasneeza Liza Zakaria (Universiti Malaysia Perlis, Malaysia), Dr. Mushtaq Ali, Dr. Abid Hussanan and Dr. Aws Naser Jaber. My appreciation also goes to FSKKP staff members, for helping me with administrative matters during these years.

On a special note, I thank my dearest parents, siblings, and friends for their endless, unconditional love and tremendous support. Finally, I would like to acknowledge research grant RDU120702 and UMP GRS GRS140392 for supporting this research.

Above all, I appreciate the almighty Allah for giving me life, wisdom, and good health to embark on this PhD journey and to complete it successfully.

ABSTRAK

Sejak kebelakangan ini, botnet telah mula tersebar dalam telefon pintar dan peranti mudah alih selepas memberi kesan kepada komputer peribadi. Botnet adalah rangkaian peranti mudah alih yang telah dijangkiti seperti telefon pintar, jam tangan pintar dan notepad, yang dikawal oleh bot-herder (botmaster). Botnet yang mensasarkan telefon pintar dan peranti mudah alih yang menggunakan sistem pengendalian Android adalah kerana ciri-ciri mereka yang sangat peribadi dan berkuasa. Akibatnya, botnet Android boleh digunakan untuk memulakan pelbagai serangan terkoordinasi yang diselaraskan termasuk e-mel spam, klik penipuan, perlombongan bitcoins, serangan distributed denial of service yang menyebarkan malware dan banyak lagi. Untuk mengesan serangan botnet yang menyebabkan kekacauan dan masalah besar kepada telefon pintar, pertamanya botnet Android perlu dianalisis. Terdapat tiga jenis analisis botnet yang terkenal iaitu statik, dinamik dan hibrid. Analisis statik mengkaji kod aplikasi dengan teliti, analisis denamik mengkaji tingkah laku aplikasi botware, sementara analisis hybrid adalah gabungan kedua-dua analisis tersebut. Walaupun analisis yang sedia ada telah memperoleh ketepatan yang baik, tetapi penyerang sentiasa mencari cara baru untuk melangkau pengesanan ketika melakukan aktiviti berbahaya. Tambahan pula teknik pengesanan sedia ada hanya dapat mengesan aplikasi Android yang berniat jahat, sementara mereka tidak dapat mengesan aplikasi botnet Android. Tujuan kajian ini adalah untuk mencadangkan pendekatan analisis statik. Dengan menggunakan teknik pembelajaran mesin untuk mengklasifikasikan botware dan aplikasi tulen. Klasifikasi ini dilakukan berdasarkan botnet yang berkaitan dengan pola unik ciri tambahan seperti keizinan, aktiviti, penerima broadcast, perkhidmatan dan panggilan API. Ciri-ciri ini dapat mendedahkan maklumat sensitif yang disimpan pada peranti mudah alih Android. Aplikasi Botware yang digunakan dalam kajian ini mengandungi 3535 sampel yang diperoleh dari dataset Contagio dan Drebin serta aplikasi tulen yang mengandungi 3500 sampel. Hasil yang diperoleh menunjukkan bahawa dengan menggunakan ciri-ciri tambahan, ketepatan pengesanan depat diperbaiki. Penilaian eksperimen berdasarkan dataset standard menunjukkan bahwa pola unik yang dipilih dapat mencapai ketepatan pengesanan yang tinggi dengan tingkat positif palsu yang rendah. Ujian eksperimen dan statistik menunjukkan bahawa ketepatan 97.28% dicapai oleh pengkelasan Random Forest machine yang berfungsi dengan baik berbanding dengan algoritma pengelasan lain. Berdasarkan hasil ujian, pelbagai isu penyelidikan terbuka yang perlu ditangani dalam kajian masa depan dapat diserlahkan.

ABSTRACT

In recent years, the botnets have started to evolve in the smartphones and other mobile devices after having an impact on the personal computers. A botnet is a network of infected mobile devices such as smartphones, smart watches, notepads, which are remotely controlled by the bot-herder (botmaster). The botnets targeting the smartphones and mobile devices which are using Android operating system due to their highly personal and powerful attributes. As a result, Android botnet can be used to initiate various distributed coordinated attacks including spam emails, click frauds, bitcoins mining, distributed denial of service attacks disseminating other malware and much more. In order to detect botnet attacks which causes immense chaos and problems to smartphones, first the Android botnet need to be analysed. There are three prominent types of botnet analyses namely static, dynamic and hybrid. Static analysis examines the application code thoroughly, dynamic analysis examines the behaviours of the botware applications, while hybrid analysis is the combination of both of these analyses. Although the existing analyses have been obtained a good accuracy, but the attackers find novel ways of skipping the detection while performing harmful activities. Furthermore, the existing detection techniques can detect only malicious Android applications, while they are unable to detect the Android botnet applications. The aim of this study is to propose a novel static analysis approach. That adopts machine learning techniques to classify botware and benign applications. This classification is performed on the base of botnet related unique patterns of additional requested features namely permissions, activities, broadcast receivers, services and API calls. These features are able to disclose the sensitive information stored on the Android mobile devices. The botware applications used in this study containing 3535 samples were obtained from the Contagio and Drebin datasets, as well as the benign applications containing 3500 samples. The obtained results show that by using the additional features the detection accuracy improved. The experimental evaluation based on real-world benchmark datasets shows that the selected unique patterns can achieve high detection accuracy with low false positive rate. The experimental and statistical tests show that 97.28% accuracy achieved by Random Forest machine classifier, it performs well as compared to other classification algorithms. Based on the test results, various open research issues which need to be addressed in future studies are highlighted.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATION

API	Application Programming Interface
APK	Application Package
APP	Application
C&C	Command and Control
CSV	Comma Separated Values
DDNS	Distributed domain name system
DDoS	Distributed Denial of Service
DOS	Denial of Service
FN	False Negative
FNR	False Negative Rate
FP	False Positive
FPR	False Positive Rate
FTP	File Transfer Protocol
GPS	Global Positioning System
GUI	Graphical User Interface
HIDS	Host-based Intrusion Detection Systems
HTML	Hypertext Mark-up Language
HTTP	Hyper Text Transfer Protocol
ICCID	Integrated Circuit Card identifier
IDS	Intrusion Detection System
IMEI	International Mobile Equipment Identity
IMSI	International Subscriber Equipment Identity
IP	Internet Protocol
IRC	Internet Relay Chat
MD5	Message Digest 5
MLP	Multilayer perceptron
MMS	Multimedia Messages Service
NB	Naïve Bayes
NFC	Near Field Communication
NIDS	Network Intrusion Detection Systems
OS	Operating System
P2P	Peer to Peer Protocol

PC	Personal Computer
RF	Random Forest
SIM	Subscriber Information Module
SLR	Simple Logistic Regression
SMS	Short Messages Service
SMTP	Simple Mail Transfer Protocol
SVM	Support vector machine
TCP	Transmission Control Protocol
TNR	True Negative Rate
TPR	True Positive Rate
UDP	User Datagram Protocol
URL	Uniform Resource Locator
Wi-Fi	Wireless Fidelity

CHAPTER 1

INTRODUCTION

1.1 Overview

This chapter describes the basis of the research work carried out in this study. The background of the initial research domain, smartphone and Android botnet is provided. This chapter is divided into nine subsections including this section. Section 1.2 highlights the background of the study and Section 1.3 describes the motivations for the research by explaining the field of research namely Smartphones, Android botnet attacks, and Android applications. Section 1.4 presents the established research problem while Section 1.5 highlights the research goal and objectives. In Section 1.6 the scope of this research study was described with the description. This is followed by the expected contribution of this research study in Section 1.7 and finally, Section 1.8 presents the thesis organization.

1.2 Background

Mobile devices such as smartphones, Personal Digital Assistants (PDA), smart-watches, and tablets have brought a massive change in the lives of people. Among them, smartphones have really changed the way of communication, due to their likelihood nature and remarkable features, particularly, telephony, multimedia, perception, and geolocation services (Analytics, 2014). As at present, these are the commonly-used tool for communication that offers users a great platform for accessing a wide range of Android applications. Therefore, these applications make the smartphones an emerging point of purchase. However, the security vulnerabilities arise from the recent infiltration of attacks on smartphones due to the increasing market penetration of mobile technology (Liao & Li, 2014). One of the security vulnerability to smartphone is botnet attacks which exploit the credential information of the end users (Wang, P. *et al.*, 2014). These attacks are posing an alarming and arguably the most potent threat to the security of Internet-connected devices such as smartphone, tablets, and smart watches (Sanz *et al.*, 2013).

A botnet is the network of infected smartphones for instance, bots that perform malware activities in a group. A bot is a type of malware that runs automatically after installation in the victim device and get the full control of that device. The botnets can be platform for a slave provided by Internet connected computers. Botnet classification is grouped into traditional and mobile botnets (Karim, A., *et al.*, 2014). In the case of the traditional botnets, the platform for the slave bots is provided by computers. However, in Android botnet, the platform for a slave bot is provided by a mobile device. Furthermore, Android botnet can be generally classified as HTTP-based, IRC-based, and P2P-based botnets according to the underlying C&C (Command and Control) communication protocol (Nigam, 2015). Unlike the traditional cybercrime, an Android botnet can attack and propagate itself through various methods and may cause much great losses to the smartphones (Guo *et al.*, 2012). In this study, mobile botnet and Android botnet have the same meaning.

Furthermore, the Android botnet could be distinguished from the other malware through its communication skill with the botmaster. It has the ability to propagate itself and launch further attacks inside the smartphones. Although the terms malware, spyware, adware, viruses and botnets are used interchangeably, yet their activities differ from one another (Paganini, 2013). The characteristics of attack vector of the Android botnets are described in a research; for example, they can attack other mobile devices via SMS, MMS, Bluetooth, and traditional IP applications protocol (Becher *et al.*, 2011). In addition, it can spread easily and quickly by combining multiple communication methods (Shin *et al.*, 2015).

In contrast, a malware is annoying, malevolent or intrusive program. As an illustration Backdoor, Trojan, and Rootkit are intended to manipulate a smartphone without knowledge of owners (La Polla *et al.*, 2013). A malware is often distributed in an infected website as a spam within a malicious link or attachment. Since, it is not required for a malware program to manipulate by a remote command and control servers. Therefore, the major difference between Android botnet and Android malware is the unconditional control of a remote machine through the Android botnet. Therefore, this study introduces the C&C- enabled Android applications as Android botnet application, and in this study, these terms will be utilized interchangeably.

In recent times, networks of mobile devices or Android botnets have become significantly involved in launching different malicious activities (Lee & Lee, 2014). For instance, Zeus as reported (Binsalleeh *et al.*, 2010) is a mobile botnet that perform malicious activities and target the end users of the Android, iOS, Blackberry, Symbian and Windows. Furthermore, it performs propagation of worms, stealing of sensitive information, accessing of the unauthorized root, spam email generation, Distributed Denial of Service (DDoS) attacks, battery outage (power consumption), processor usage, and memory consumption (Narudin *et al.*, 2016). In addition, as Android botnet operations can be implemented by distributing malicious applications to mobile subscribers, both concepts are therefore interrelated (Karim, A. *et al.*, 2015). Conclusively, Android botnet is one of the most serious threats to the smartphone users.

The state-of-the-art literature shows that a large number of detection techniques have been proposed for Android botnet attacks. These techniques are fundamentally broken into static, dynamic, and hybrid detection techniques (Peiravian & Zhu, 2013; Sanz *et al*., 2013). In the static detection technique, the static features namely permissions and API calls of Android applications are analysed. On the other hand, the dynamic features in dynamic detection technique include battery, memory, and network utilization are analysed in the runtime behaviours of the smartphone's applications. Finally, the hybrid detection technique is the combination of static and dynamic techniques as described above.

1.3 Motivation

Smartphone is a rapidly growing technology in terms of both research work and commercial applications. Over the last few years, smartphone has grown exponentially from its origin to the existing vast research and applications development industry. The smartphone 'mobility' was predicted in Ericsson's report to have 2.9 billion subscription by the end of 2016 (Ericsson, 2016). In a related report, the overall number of smartphone users reached to 3.10 billion by the end of 2016 (Statista, 2016). Despite the characteristics, such as telephony, multimedia, perception, and geolocation services, smartphone is vulnerable to botnet attacks because of its easy accessibility and distributed infrastructure (Kirubavathi & Anitha, 2017). In spite of this threat to smartphone, the users of the smartphone are increasing rapidly (Woods, 2016). Smartphones have a rich-featured operating system (OS), integrated with powerful hardware such as Android, iOS, Blackberry, and Windows Phones (Gilbert, April 2012; Microsoft-Inc., 2017; Rubin, 2008). Figure 1.1 shows the number of smartphone users with respect to the installed OSs. It shows that Android OS have more number of users as compared to other OSs.

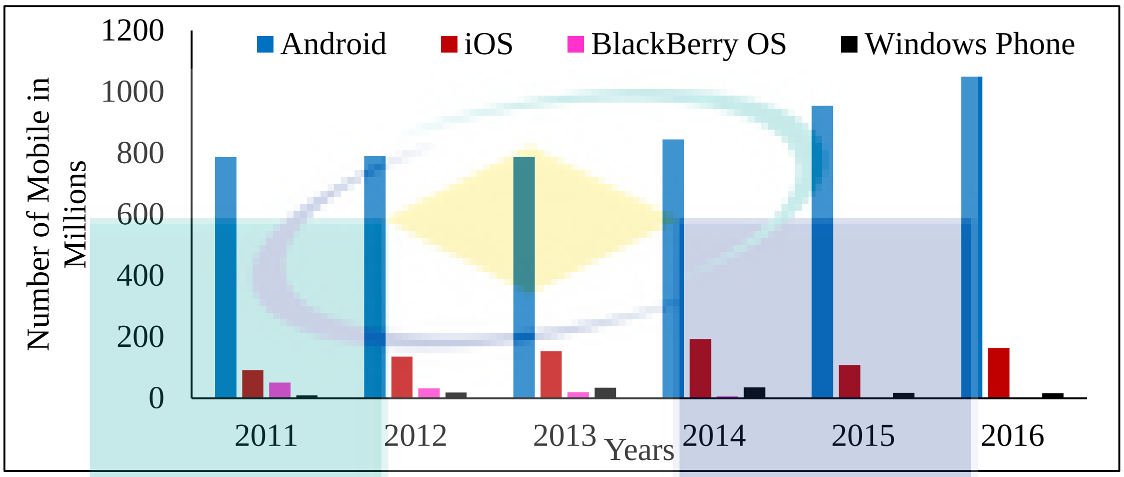


Figure 1.1 Number of Smartphone Users on the Basis of Operating System

Source: Statista (2016)

Android OS is one of the dominant platform commonly used by mobile devices due to its incredible traction with an extensive range of users (Suarez-Tangil *et al.*, 2014). Consequently, smartphones as the primary choice of computing device have replaced the personal computers (PCs). It is clear from the current statistics that since 2011 the global shipment of smartphones has increased as compared to the PCs (Karim, A. *et al.*, 2015). Resultantly, the deployment of 4G technology such as WiMAX and LTE would become the main source of Internet access in the near future.

Admittedly, cybercriminals have been motivated by this technological shipment to exploit the vulnerabilities of smartphones through off-the-shelf malware creation tools (Ollmann, 2009). Moreover, the worldwide availability of the Android applications through the Internet spread the malicious code to the smartphone users. Currently, Android botnet attacks are the most evolving trend in the malicious code (Kirubavathi & Anitha, 2017). These reports prove that the effect of Android botnet attack is unavoidable and has the ability to gain full control of the smartphone and its contents. This development led the researchers to explore the Android botnet attacks in the smartphones.

Consequently, this study is carried out to help C&C communication pattern in Android applications and ultimately increase the probability of detecting botnet attacks in Android smartphones. In order to detect Android botnet attacks, three of the common analysis approaches namely static, dynamic, and hybrid are used (Peiravian & Zhu, 2013; Sanz *et al*., 2013). In static analysis approach, it does not require the execution of malicious program code. In this approach, the static features of the applications are extracted by disassembling the program code such as permissions, API calls, explicit and implicit features (Feizollah *et al.*, 2015). Therefore, static analysis is known as a lightweight detection mechanism botnet analysis. However, it is unable in the static analysis to illustrate the behaviour of the program completely (Stuvert & Soniya, 2015). Conversely, in the dynamic analysis it requires the execution of malicious applications in a virtual environment called the sandbox to monitor the runtime traces and to extract their dynamic features i.e., System calls. While the hybrid analysis first run the static approach for static features analysis and then dynamic approach for behaviour analysis (Shi *et al.*, 2016). Although, the existing botnet detection approaches are facing some limitations, such as, dynamic approach needs more powerful deployment machine for execution of each botware application in an isolated environment (Karim, Ahmad *et al.*, 2015). Similarly, hybrid approach needs to extract the static features from the botware applications and then execute for dynamic features extraction in an isolated environment.

As mentioned in the previous section, only C&C server differentiates the mobile botnet from the mobile malware. This has resulted into existing detection mechanisms to focus on mobile malware at a large scale. Moreover, most of the existing mobile attacks replicate the nature of PCs based attacks. Therefore, many of the existing detection solutions can also be applied to the mobile threats (Faruki *et al.*, 2015). Notwithstanding, smartphones namely Android, iOS, and Windows have their own constraints due to their limited resources namely battery consumptions, reduced processing, heterogeneity and low data storage capabilities (Penning *et al.*, 2014). Hence, these limitations restrict the detection mechanism to be efficiently programmed.

1.4 Problem Statement

Since the Android operating system got popularity in the last few years. With the passage of time, the cyber attackers start developing botnet applications when they saw Android OS as a flourishing target. The growth of Android botnet applications increased, in term of complexity and volume as it has some significant advantages over traditional botnets since smartphones are rarely either switched off, or disconnect with the Internet which makes it more reliable (Anagnostopoulos *et al*., 2016).

Many researchers from academia and industries have proposed botnet analysis and detection approaches (Faruki *et al.*, 2015; Peng *et al.*, 2014). However, the cyber-attackers have always tried to find a way to evade new detection approaches, since botware are still exist inside the store (PlayStore, 2017). Thus the existing analysis and detection approaches are unable to detect mobile applications that are involved in botnet activities such as information hijacking, remote access, DDoS, phishing, and perform action according to the botmaster instruction to launch and initiate a botnet attack (Johnson & Traore, 2015; Karim, Salleh, & Khan, 2016; Sanz *et al.*, 2013). Moreover, analysis on many botnet applications with diverse features set is a challenging task, and as a result of that, it is crucial to select the exact static and dynamic features with botnet capabilities (Kazdagli *et al.*, 2016).

The static analysis and detection approach are the commonest type in the existing botnet analysis techniques but failed to provide the absolute picture of the android applications behavior with the existing static features such as permissions and API Calls (Peiravian & Zhu, 2013; Rashidi & Fung, 2016; Sanz *et al.*, 2013; Yerima *et al.*, 2014a). In contrast, dynamic analysis detection techniques need computation intensive resources processing time and code coverage (Fan *et al.*, 2017; Spreitzenbarth *et al.*, 2015). Furthermore, the existing botnet detection techniques in smartphones are unable to detect the botnet attacks based on these two selected features.

Thus, it is necessary to propose a new analysis and detection approach based on additional features such as activities, broadcast receivers, and services. Consequently, there is a degrade in the accuracy and TPR due to the selection of inefficient and irrelevant features set. Therefore, the proposed analysis and detection approach can be enhanced by adding the feature refinement process to improve the detection efficiency that leads to increase in the accuracy, true positive rate (TPR), precision, F-Measure and decrease the false positive rate (FPR).

1.5 Research Goal and Objectives

This research is undertaken with the aim to propose an enhanced android botnet detection approach using feature refinement. The aim of this research is accomplished by addressing the following objectives.

1. To examine the characteristics of botnet attacks in the smartphone and Android applications in order to derive a concise set of features that are effective for Botnet detection.
2. To design an improved botnet detection approach that has features refining component for observation and detection of static features of Android applications with botnet capabilities.
3. To evaluate and validate the performance of the proposed approach by considering five matrices including: TPR, FPR, precision, F-measure and accuracy and compare it with state-of-the-art Android botnet detection techniques.

1.6 Scope

The study undertaken in this thesis is aimed to propose an enhance approach by using the feature refinement for Android botnet detection in mobile devices. This study is limited to Android operating system and Android applications that are available from third party developers. Furthermore, this study focused on the static features of an Android application such as permissions, activities, broadcast receivers, services, and API calls. However, it is limited to device-based Android botnet detection rather than network-based. Figure 1.2 shows the scope of this study diagrammatically.

This proposed approach is based on the static analysis approach. Although, static analysis approach is 1well-known in botnet detection but recently gained popularity as an efficient mechanism for smartphone protection. This approach is a relatively fast and it has been widely used in malware analysis to search for suspicious strings or blocks of code.

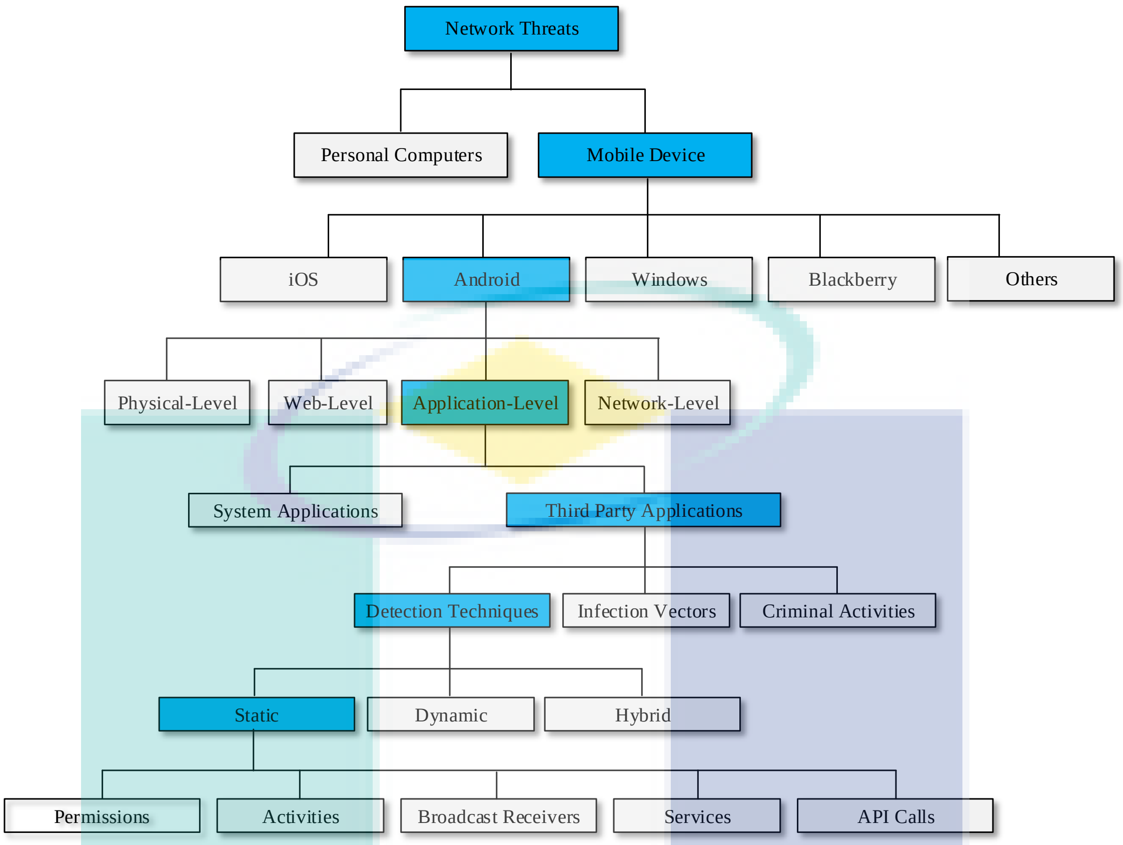


Figure 1.2 Diagrammatically representation of Scope of this Study

1.7 Expected Contribution

This Section highlighted the contribution of the research study undertaken in this thesis. The expected main contribution of this research is the enhancement of botnet detection in Android devices using feature refinement. Other expected contributions to the body of knowledge are as follow:

- A comprehensive taxonomy representing the Android botnet attacks will be proposed. Moreover, state-of-the-art Android botnets are investigated according to the proposed taxonomy.
- An enhanced approach will be proposed for Android botnet detection using feature refinement. This has the capabilities to effectively detect botnet C&C communication features in Android smartphones by investigating the manifest.xml and DEX files.
- Proposed a novel feature refining approach which address the most prominent features in order to inspect the range of their frequencies in Android applications.

- Generation of the analytical evaluation results for the proposed approach through Drebin dataset was made possible. Currently, Drebin is the largest available dataset for Android botware. In addition, performance evaluation on the unmodified (without feature refining) and modified (feature refining component) approaches will be carried out. Lastly, a statistical model for the evaluation parameters of proposed approach and detection of Android botnet attacks will be developed.

1.8 Thesis Organisation

The research entitled an enhanced android botnet detection approach using feature refinement is an emerging field that involves an extensive study. In view of this, the thesis has been arranged into five different chapters for clear understandability of the readers with a layout shown in Figure 1.3.

Chapter 2 aims to review the research undertaken in the field of botnet attacks and their detection in smartphones. The chapter describes the knowledge about the mobile devices (smartphones) and the vulnerability of the botnet attacks in order to identify and classify these botnet attacks in smartphones. Moreover, in this chapter mobile botnet attacks are focused on, and the details about the detection techniques for these attacks are provided which discover the deficiency of the existing detection techniques. The detection techniques are categorized into main three categories such as static, dynamic and hybrid detection techniques. Furthermore, each category is given in detail. Qualitative critical analysis in the aforementioned research direction was provided based on the metrics derived from the proposed taxonomy. Hence, the research problem is identified as developing detection techniques in smartphones based on static analysis.

Chapter 3 outlines the enhanced botnet detection approach using feature refinement component. incorporates additional refining component and features proposed approach. By using schematic presentation, the major components of the proposed approach and their functionality are explained in more details. It discusses various components of the proposed approach along with their functions. In addition, methods and services used in the proposed approach are explained in detail.

Chapter 4 describes the experimental setup, tools. The results obtained from the experiments are summarized in this chapter. The experimental setup is described with accompanying datasets and devices. The data collection method and evaluation methods namely statistical modelling and datasets that have been utilized to evaluate and validate the proposed approach performance are adequately described. The experimental evaluation is based on the five metrics, namely TPR, FPR, precision, F-measure, and accuracy. On the long run, the result based on these evaluations metrics is used to prove the efficiency and significance of proposed approach. The next section summarizes the performance evaluation methodology for the proposed approach on collected data in order to highlight the strengths and weaknesses of this approach.

Finally, **Chapter 5** concludes this work by showing accomplishment of the aim and objectives of this study. The contribution of the research is summarised in this chapter. The open research issues and future research directions concluded at the end of Chapter 5.

In addition, there are number of appendices included at the end of this study. The list of published journal and conference articles related to the study undertaken in this thesis are given in **Appendix A**. Appendix B includes the number of features with their indexes, Appendix C contains the unique patterns generated from the features. Furthermore, the list of experimental results are given in Appendix D.

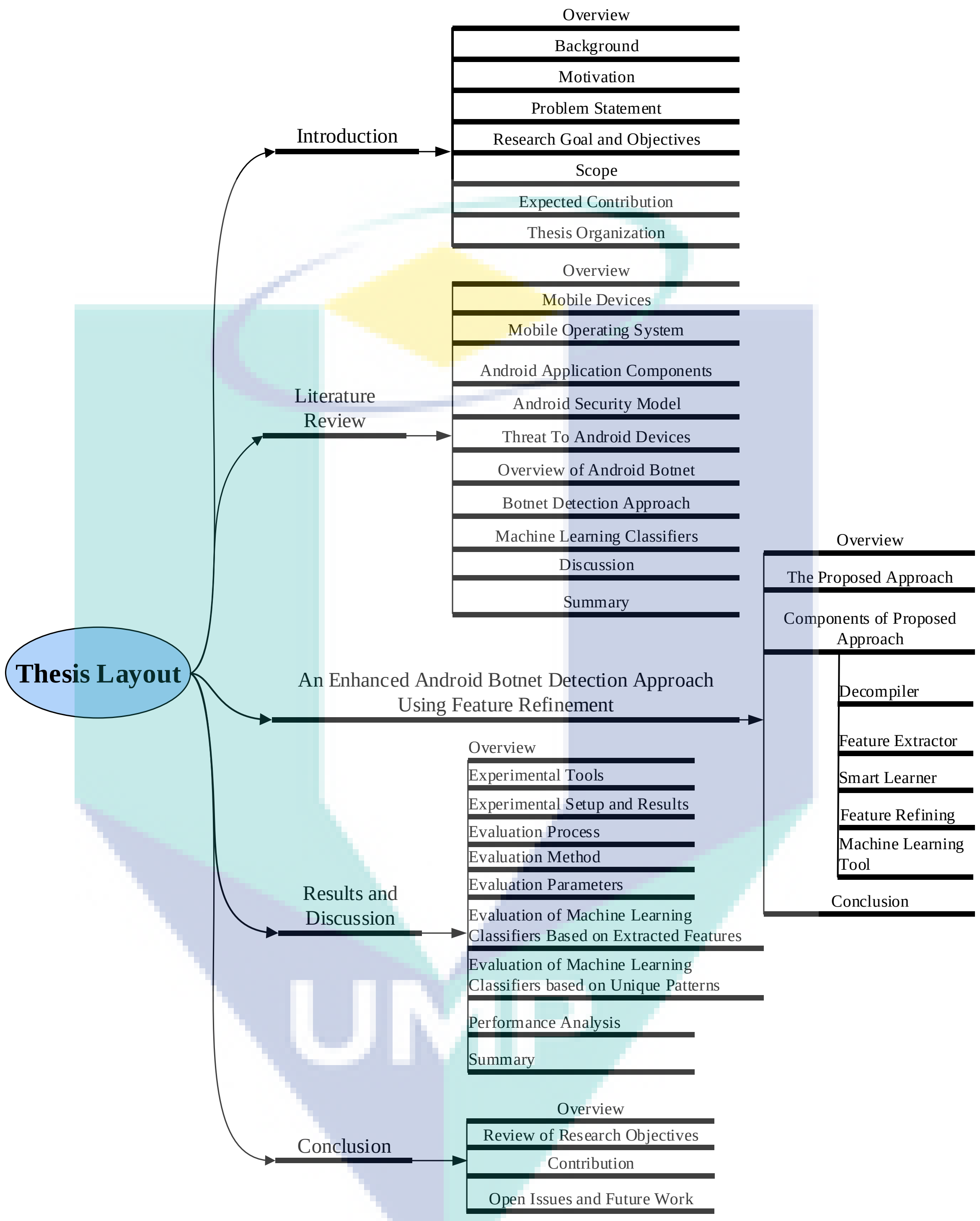


Figure 1.3 Thesis Layout

CHAPTER 2

LITERATURE REVIEW

2.1 Overview

This chapter describes the mobile devices, the phenomenon of botnet in mobile devices such as smartphone, tablets, smartwatches. The threats to mobile device are classified in this chapter. While the chapter is divided into nine sub-sections, Section 2.2 reported mobile devices in detail. Again, Section 2.3 highlights the mobile operating systems. Section 2.4 describes the components of the Android application. Section 2.5 given detail about the Android security model followed by threats to Android devices in Section 2.6. The overview of Android botnet is given in Section 2.7. The existing botnet detection approaches are given in Section 2.8. Furthermore, machine learning classifiers are given in in Section 2.9 and discussion is given in Section 2.10. In the last Section 2.11 summarises the whole chapter.

2.2 Mobile Devices

A mobile device is a small enough to the handheld computing device, with input and output capability. Mobile devices are performing a progressively essential role in the current time (Suarez Tangil *et al.*, 2014). There is an established fact about great capabilities and exploits of the mobile devices in today's technology. The number of mobile devices precisely related to the facilities they provide to the end-users, showing that they will very nearly outsell the number of PCs worldwide (Feizollah *et al.*, 2014; Tam *et al.*, 2017). Each mobile device can run diverse types of applications software (AppStore, 2017; Othman *et al.*, 2014; PlayStore, 2017). According to Dinh *et al.*, (2013*)* smartphones, tablets, smart watches, and personal digital assistants are the common types of mobile devices. Figure 2.1 illustrates the number of mobile devices users in billion as per 2018. As reported by GSMA Intelligence, that currently there are 7.22 billion mobile devices in the world. However, according to the US Census Bureau this number is still between 7.19 and 7.2 billion. (Miakotko, 2017)

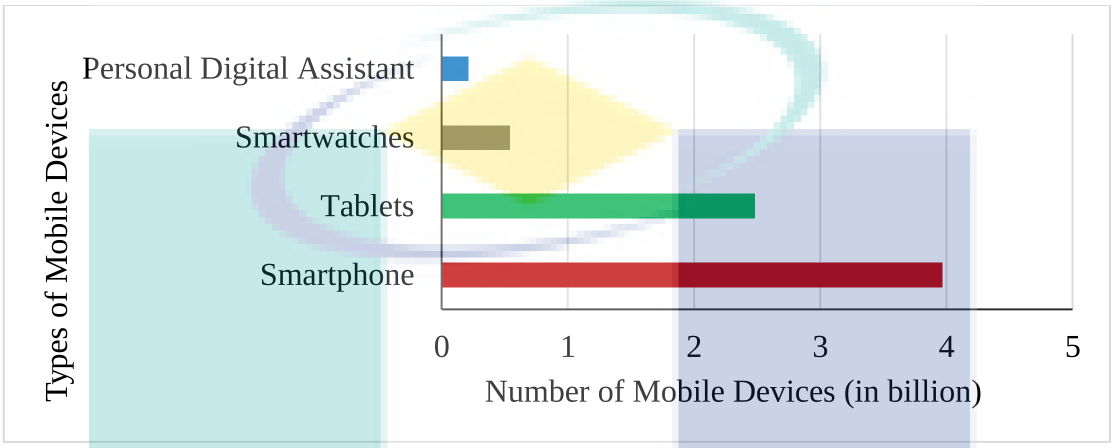


Figure 2.1 Number of Mobile Devices (in Billion)

Smartphone is a communication device like a cell phone having touchscreen interface and some advanced functionality alongside with making phone calls and sending messages (Zonouz *et al.*, 2013). According to Rubin "There should be nothing that users can access on their desktop that they cannot access on their smartphone" (Rubin, 2008). It is like a personal computer in small size, having the capabilities of place and receive calls (Zhou & Jiang, 2012). Moreover, a smartphone is more than just a cell phone; it is a media player, a gaming console, a camera, a video recorder, a document editor, and a GPS navigational device. Every smartphone has a rich-featured operating system, integrated with powerful hardware.

2.3 Mobile Operating System

Operating system is a system software that runs on mobile devices to control and run the system hardware. These OS has an openness nature and rich functionalities which provides a platform for other applications software to run on mobile devices (Dai, Q. *et al.*, 2012). Figure 2.2 shows the usage percentage of existing top three mobile operating systems namely Android, iOS, and Windows (AppStore, 2017; Microsoft-Inc., 2017; Tam *et al.*, 2017; Techopedia, 2017).

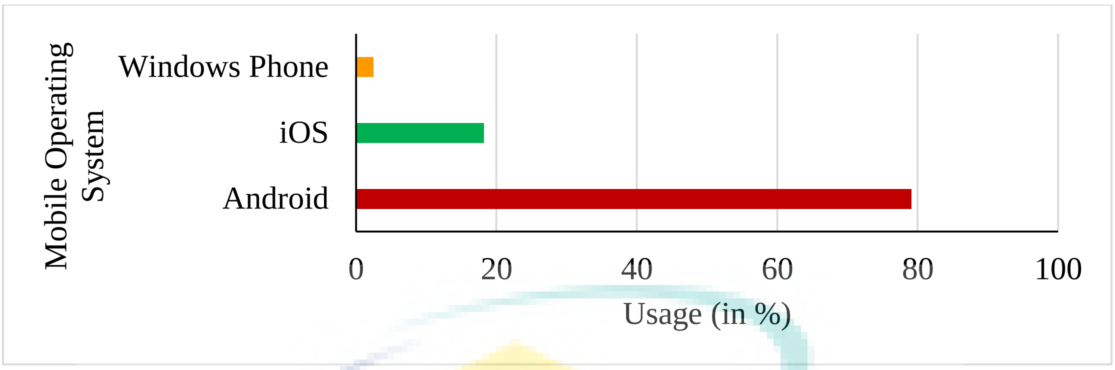


Figure 2.2 Mobile Operating System Usage in Percentage

2.3.1 Android Operating System

Android is a Linux-based operating system, used by the aforementioned mobile devices, developed by Google in conjunction with the Open Handset Alliance (Barrera & Van Oorschot, 2011). It provides a complete set of software for smartphone devices including operating system, middleware, and key mobile applications (Sears, 2007). It is the most prominent operating system in the recent time due to its open nature, and fewer control on third parties application distribution system (Zaman *et al.*, 2015). According to a report Android's market share accounts for over 82% with 1.4 billion Android users (Statista, 2016). It allows the users to download and install these applications from an untrusted source due to its open nature (Peiravian & Zhu, 2013).

Android can be thought of as a software stack comprising different layers, each layer manifesting well-defined behavior and providing specific services to the layer above it. Android uses the Linux kernel, which is at the bottom of the stack. It enables Android to support a vast array of devices, and it makes it easy for developers to write drivers in a well-understood way (Techopedia, 2017). Runtime Dalivik VM and Core libraries are on the top of the Linux kernel. These libraries includes a set of C and C++ libraries used by different components of the Android OS. Developers use these libraries through the Android application framework. Application framework are built on top of the Libraries, which makes able the Android applications to interact with the Kernel and Libraries. Android application is the topmost layer of the Android OS. Figure 2.3 depicts the complete "Overview of the Android Operating System Architecture".

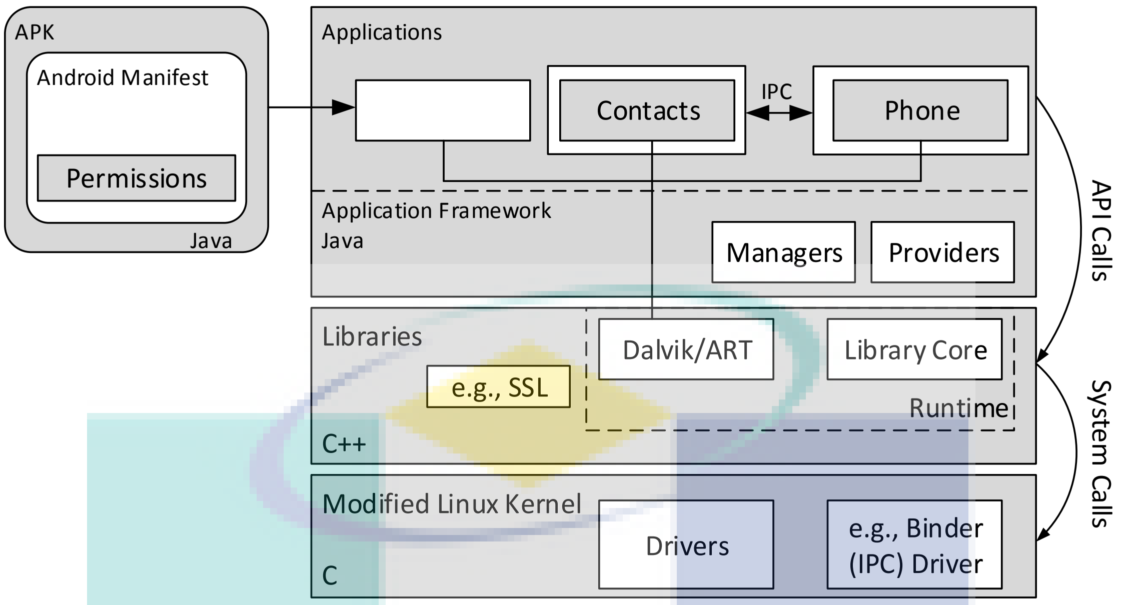


Figure 2.3 Overview of the Android Operating System Architecture

2.3.2 Windows Operating System

Windows mobile OS is an operating system developed for mobile devices by Microsoft based on the Windows. This OS is resemblance to desktop versions of Microsoft Windows. Initially windows is debuted as the OS for Microsoft's original personal digital assistant (PDA) device in 2000. However, with the passage of time and popularity Microsoft designed a windows OS for enterprise handheld (mobile) devices (Microsoft-Inc., 2017). The highly sensitive review ability of the apps makes the windows OS does not need for a dedicated anti-virus software (Salah *et al.*, 2013). Moreover, the applications for Windows mobile OS are available to purchase from the Windows marketplace.

2.3.3 iOS

iOS is a mobile operating system runs on Apple devices such as iPhone, iPod, and iPad. iOS is developed by Apple’s Inc (Apple-Inc., 2017). The performance of the iOS is better than other OS, but there are some disadvantages of iOS such as it is not flexible, it is not open source, the available applications are very expensive (Suarez Tangil *et al.*, 2014). Apple’s App Store is a protected market with an uncompromising process of review. The strict rules of iOS for developing the applications restrict the developers but still there are over 2 million iOS apps available for download in the Apple App Store.

2.4 Android Application Components

A typical Android application is a package in an APK file (Android applications package) and usually rich in functionality. Each android application must have these components namely activities, services, broadcast receivers, and content providers. These are the essential part of each application. The APK files contains the compiled Java code and other resources like texts and images for the Android application

2.4.1 Activity

An activity provides a screen to interact with and defines the interaction sequences and UI layout presented to the user (PlayStore, 2017). Most applications will have multiple activities (one for each screen that the user sees/interacts with). The user will switch back and forth among activities (in no particular order, and times). Activities have to be registered in the manifest and cannot be added programmatically. For example, Figure 2.4 shows the basic structure of an activity declaration in AndroidManifest.xml file.

```
<activity>
     <intent-filter>
          <action />
          <category />
          <data />
     </intent-filter>
     <meta-data />
</activity>
<activity-alias>
     <intent-filter> . . . </intent-filter>
     <meta-data />
</activity-alias>
```

Figure 2.4 Structure of an Activity declaration in AndroidManifest .xml file

Figure 2.5 shows the life cycle of an Android applications activity component which can be modeled by a state machine. It describes the state-dependent behaviour of an Android activity. This describes different phases of the activity component by the states, while the transitions within each phase are illustrated by the state transitions. This component contains seven states namely, created, killed, started, stopped, running, paused, and destroyed. In this model, each state transition is caused by the calling of an

OS callback. However, these callbacks are called callbacks life cycle. For example, when an end user starts or opens an activity, the event will generate by createActivity as shown in the first operation. The system invokes the callback sequence (e.g., onPause (), onCreate (), onStop (), onStart (), onResume (), and onDestroy ()) in the activity and in response, the activity makes transitions to Created, Stopped, Killed, Destroyed, Started and Stopped state, respectively (Junaid *et al.*, 2016). Activity waits for user interaction in the stopped state. An end user can generate many events in this state that can make to visit different state by an activity.

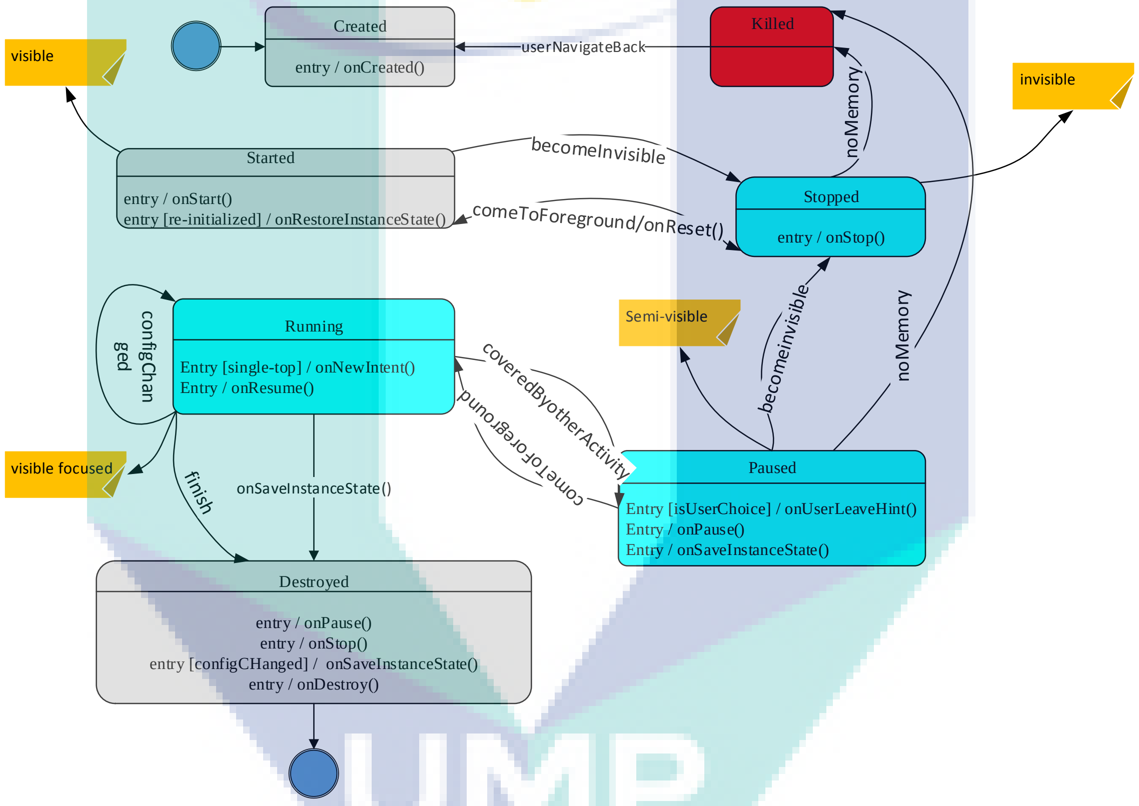


Figure 2.5 Android Application Activity Lifecycle

Source PlayStore (2017)

2.4.2 Service

This is the basic component of an Android application that can perform its operations in the background for Android application. It does not have a user interface component, but it performs its duty in the background and executes its tasks, for example, a music player, time, and alarm. All the Android applications are running in the front, while the services are always active behind the curtain. It does not affect the services, even if the user switches to other applications. In addition, an application component may "bound" itself to a service and thus interact with it in the background; for example, an application component can bind itself to a music player service and interact with it as needed. Thus, service can be in two states such as: Started or Bound.

When an application component launches a service, it is "started." This is done through the startService() callback method. Once the service is started, it can continue to run in the background after the starting component (or its application) is no longer executing. An application component can bind itself to a service by calling bindService(). A bound service can be used as a client-server mechanism, and a component can interact with the service. The service will run only as long as the component is bound to it. Once it unbinds, the service is destroyed. Any application component (or other applications) can start or bind to a service once it receives the requisite permissions. This is achieved through Intents. Creating an application service requires that one must create a subclass of service and implement callback methods. Most important callback methods for service are onStartcommand(), onBind(), onCreate(), and onDestroy(). Figure 2.6 represents the basic structure of service in the Android application.

```
<manifest ... >
  ...
  <application ... >
      <service android:name=".ExampleService" />
      ...
  </application>
</manifest>
```

Figure 2.6 Structure of Service declaration in AndroidManifest .xml file

2.4.3 Broadcast Receiver

Broadcast receiver component is an asynchronous event mailbox for Intent messages "broadcasted" to an action string. Most of the executing malicious Android applications, is observed to listen to broadcast receivers, such like SEND_MESSAGE, BOOT_COMPLETED, OUTGOING_CALL, SMS_RECEIVED and much more (Alazab et al., 2012). Android defines many standard action strings corresponding to system events, such as LOW_BATTERY_NOTIFICATION, LOW_MEMORY and much more. Both benign and malicious applications can also end broadcast messages as 'intent messages' to the system; for example, indicating that applications are waiting for an event. Broadcast receiver attackers can design their malicious application to listen for incoming messages and forward them to predetermined or premium numbers, likewise, developers often define their own action strings. Figure 2.7 shows the basic structure of broadcast receiver.

```
<receiver android:name=".MyBroadcastReceiver"    android:exported="true">
   <intent-filter>
          <action android:name="android.intent.action.BOOT_COMPLETED"/>
          <action android:name="android.intent.action.INPUT_METHOD_CHANGED" />
   </intent-filter>
</receiver>
```

Figure 2.7 Structure of Broadcast Receiver declaration in AndroidManifest .xml file

2.4.4 Content Provider

A content provider component provides data from one application to others on request. The content provider needs to be declared like other application components in the Manifest.xml file. It can examine who can access the content provider by defining permissions inside the <provider> tag. It provides an intent which enable sharing of data between two applications. These requests can be handled by the Conten.Resolver.query(), Content.Resolver.insert() methods of the Content Resolver class. The request will be denied if the caller does not have a proper permission. The data may be stored in the file system, the database or somewhere else entirely. A content provider is implemented as a subclass of Content Provider class and must implement a standard set of APIs that enable other applications to perform transactions.

2.5 Android Security Model

Android developers have included security in the design of the platform itself. This is visible in the two-tiered security model used by Android applications and enforced by Android. Android, at its core, relies on one of the security features provided by Linux kernel- running each application as a separate process with its own set of data structures and preventing other process from interfering with its execution.

Android allow the applications or components to interact with each other’s by using the fine-grained permission at the applications layer. Once the application installed on the user device, an approval from the user is necessary to access the critical operations. The permissions are used in order to execute these critical operations successfully. By default, the third-party applications do not have any permissions to perform any critical activities that might resultantly affect the contents of other applications. Such as sending or reading SMS or MMS messages, dialling calls or accessing contact information. Permissions are categorized in application level and manifest level permissions (Moonsamy *et al.*, 2014). Application-level permissions provide a way to get access to restricted content and APIs. These permissions Access to low-level Linux facilities is provided through user and group ID enforcement, whereas additional fine-grained security features are provided through Manifest permissions.

2.5.1 Manifest Permissions

Android applications are sandboxed which imply that they are limited to use their own files and any open-accessible resources on the mobile devices. This limitation makes these devices uninteresting. Although, an Android can grant extra, fine-grained access right to these third-party's applications to permit for wealthier functionality. Those access rights are called permissions, and they can control full access to device hardware and software. These accesses include, Internet connectivity, operating system services, access external or internal data storage and much more. Permissions can be assigned to broadcast receivers, content providers, activities, and services. For example, when an Android application requests permission to access the Internet, it is essentially seeking permission to open the IPv4 and IPv6 sockets. Application permissions are then mapped to the "inet" group name through the /system/etc/permissions/platform.xml. Figure 2.8 presents the XML maps of the application's permission.

```
<uses-permission android:name= “android.permission.INTERNET”/>
<uses-permission android:name = “android.permission.NFC/”>
< uses-permission android:name = “android.permission.SEND_SMS/”>
< uses-permission android:name = “android.permission.BLUETOOTH/”>
<group gid= “inet” />
</permission>
```

Figure 2.8 An example of Permissions declaration in an application

There are several strings to declare these permissions for the usage of different Android applications with a special template as

<uses-permissions Android: name = "Android.permission. String" >.

In this section, the permissions that exist in input Android application are listed in a separate file. It shows that the INTERNET is one of the most required permissions in both botware and benign. Also, READ_PHONE_STATE, RECEIVE_BOOT_C-OMPLETED and SEND_SMS are the second, third and fourth permissions respectively that are mostly used by the benign and botware. This process is performed for all input Android applications for the purpose of decompression and extraction of used permission features, by listing them in a separate file. From the permissions analysis used by botware and benign, it is noticed that the botware have more permissions intensive as compared to the benign applications. Table 2.1 shows the top 20 permissions used by botware and benign applications. However, the Table 2.2 shows the used permissions with their description.

Table 2.1 Example of Top 20 Used Permissions by botware and benign applications

Permissions	Botware	Benign
INTERNET	97.860	51.430
READ_PHONE_STATE	95.710	31.430
READ_CONTACTS	81.430	23.570
SEND_SMS	80.710	15.000
READ_SMS	77.140	7.860
RECEIVE_SMS	71.430	5.710
CALL_PHONE	66.430	8.570
WRITE_SMS	65.000	6.430
WRITE_SETTINGS	62.140	9.290
WRITE_CONTACTS	58.570	2.860
CHANGE_WIFI_STATE	50.710	17.860
ACCESS_FINE_LOCATION	47.860	4.290
SYSTEM_ALERT_WINDOW	47.140	10.000
GET TASKS	40.000	3.570
DISABLE_KEYGUARD	31.430	11.430
ACCESS_COARSE_LOCATION	30.710	5.710
CAMERA	22.140	1.430
BLUETOOTH	20.000	2.860
PROCESS OUTGOING CALLS	20.000	2.860
RECORD_AUDIO	18.570	4.290

Table 2.2 Permissions with their Description

Permission	Description
INTERNET	This permission allows an application to Open Network socket, or call function java.net.URL-.openConnection
READ_SMS	Allow an application to read SMS on device,
READ_CONTACTS	Allows an application to read the user's contact information. These information can further propagate to the command and control server for criminal activities, like infection, stealing.

Table 2.2 Continued

Permission	Description
READ_HISTORY_B OOKMARKS	Allow an application to read the phone history bookmarks
READ_LOGS	This is a read only permission which grant permission to an Android application to read low-level system log files.
READ_SYNC_SETT INGS	By call the Android.permission.READ_SYNC_SETTINGS an application can read the sync settings.
READ_SYNC_STAT S	By calling the method Android.permission.READ_SYNC_STATS allowing to an application to read the sync stats for an account, it can read history and amount of data which is synced.
ACCOUNT_MANAGER	Allows applications to call into Account Authenticators. Using this method an application can access password, user data, and token as well
ACCESS_FINE_LOCATION GPS_PROVIDER;	By using the ACCESS_FINE_LOCATION an application can access the precise location sources, such as Global Positioning System (GPS).
READ_PHONE_STATE	Call functions getDeviceSoftwareVersion, getSubscriberId, getDeviceID, or getSimSerialNumber from the Android.telephony.TelephonyManager class or Binder transaction to com.Android.internal.telephony.IPhoneSubInfo.getDeviceId,
GET_PACKAGE_S IZE	It allows an application to find out the space used by any package
VIBRATE	It granted the permission to the Android phone vibration to an application.
SEND_SMS	It allows an application to send a single or multiple SMS from the device having an application with enabling this permission to one or more recipients or groups.
BLUETOOTH_ADMIN	By calling this function, it allows the applications to discover and pair the Bluetooth enabled devices nearby.
DELETE_PACKAGES	This is the permission which is not using for third party applications, it can allows an application to delete packages.
DELETE_CACHE_ FILES	Allows an application to delete cache files.
UPDATE_DEVICE _STATS	Allows an application to update device statistics
ACCESS_DOWNL OAD_MANAGER	Allows an application to access the download manager, edit, delete, copy or move data.
INSTALL_PACKA GES	It allows an application to install packages, while this permission is not for use by third-party applications.
RECEIVE_BOOT_ COMPLETED	It allows an application to receive the ACTION_BOOT_COM-PLETED which broadcast after the system finishes booting.

Source PlayStore, (2017); Sanz et al., (2013) Furthermore, the used permissions can be categorized into four subgroups according to their behaviour, which are Normal, Dangerous, Signature and SignatureORSystem (PlayStore, 2017). All the Android applications can request these permissions be defining them in the AndroidManifest.xml file. However, the normal permissions can be requested and used by botware and benign applications almost in equal numbers (Felt *et al.*, 2012). While the dangerous permission, are requested by botware applications in more numbers as compared to benign applications (Aswini & Vinod, 2014).

Additionally, the most prominent permissions used by botware applications are INTERNET, READ_PHONE_STATE, WRITE_EXTERNAL_STORAGE, RECEIVE_BOOT_COMPLETED, SEND_SMS, WRITE_SETTINGS, WRITE_CO-NTACTS, READ_SMS, READ_SOCIAL_STREAM, WAKE_LOCK, RECEIVE_SMS, READ_CONTACTS, ACCESS_WIFI_STATE, VIBRATE, CALL_P-HONE, WRITE_SMS among many others. Most of the botnets are using these permissions to establish a remote connection with command and control server in order to check the status of the infected devices.

2.5.2 API Calls

The Android platform provides a framework known as Application Programming Interface (API) that Apps can use to interact with the underlying Android system. The framework API consists of a core set of packages and classes. Since, most Apps use a large number of APIs, this motivates the use of API calls for each application as a feature to characterize and differentiate malware from benign Apps. This goal can be achieved, by creating a framework to reverse engineered APK file and extract API calls of each application. An example of extracted API calls is shown in the Figure 2.9.

Connect, getActiveNetworkInfo, getContent, getDeviceId, getInputStream, getLastKnownLocation, getLine1Number, getNetworkInfo, getSimSerialNumber, getSubscriberId, getWifiState, LocationListener, openFileDescriptor, requestLocationUpdates, sendTextMessage

Figure 2.9 Example of used API calls by Botware and Benign Applications

2.6 Threats to Android Devices

Each new day comes with new challenges and threats to Android devices. The same phenomenon goes for viruses and spyware which can infect PC; and there are different types of threats to mobile devices that can badly affect the mobile devices (Cole, 2012; Datar, 2013). These threats can broadly divided into four categories namely application-level threats, web-level threats, network-level threats, and physical threats (Lodi *et al.*, 2014; Ruggiero & Foote, 2011). Application-level threats are based on the Android applications, which is the core feature of every mobile device. The threat that appeared to be the most widely discussed in the literature is the application-level threat (Li, Y. *et al.*, 2014). As all the applications that run on the smartphone are available from third party markets, it is clear that these applications can be target vectors to mobile device security (Faruki *et al.*, 2015; Google, 2015). These applications that perform malicious activities are known as malware. It can injects malicious code into the Android applications to send unsolicited messages; allow an adversary the ability to remotely control the device (Narudin *et al.*, 2016). Karim. *et al.*, (2015) explained the common types of application level threats to smartphone are Adware, Ransomware, and Botnet.

2.6.1 Adware

Adware is a type of malware that sends different types of advertisement to the mobile users while using Android applications. Advertisement is the main pillar of Internet revenue model. These ads are sends and appears through popup windows on the program user interface (UI). However, sometimes these advertisements are annoying and disturbing the users. Due to geographic location, NFC, and Bluetooth communication the adware is more potential on mobile devices rather than PC counterparts. Apart from sending advertisements in mobile devices without the approval of user's, they can modify Internet browser settings, edit home screen of mobile device, and in some cases collecting credential of end users. The most alarming threat about the adware is that is these are unable to detect by the existing anti-viruses (Virustotal, 2017).

2.6.2 Ransomware

This is a new type of malware that hostage the mobile device for some financial benefits. The main motive of such malware attacks is always monetary gain. Initially the first ransomware was appeared in 2013 and it is detected in the same year, namely simplocker (Zavarsky & Lindskog, 2016). This type of ransomware targets the smartwatches. Furthermore, Android Defender was found a fake application for security which locks the mobile devices and later on demands for some finance in order to unlock the device (Symantec, 2014b). The main types of ransomware are lock-screen ransomware and crypto-ransomware. The first one lock the screen of the attacked device and stop the users from accessing their own device. While the crypto-ransomware steals the files from the devices. However, the motive of the attacker is same in both methods that is demands ransom (ESET, 2016).

2.6.3 Botnet

Botnet is short-form for robot network, which is the network of Internet connected infected devices (bots) under the control of bot-master (cyber-criminal) to perform cyber-criminal activities without knowing to the device owner (Datar, 2013; Silva *et al.*, 2013). Botnet is in two different categories which are traditional botnets and Android botnets, with the focus of this study on the Android (mobile) botnets. The aim of Android botnets will most likely be similar to those of existing traditional botnet (e.g., providing means of DoS, DDoS and spam distribution); however, the targets will change (Enck, W. *et al.*, 2009; Pieterse & Olivier, 2012). The platform is the key difference between traditional and Android botnets. In Android botnets, the victim and the attacker platforms are provided by Android smartphone which is contrary to the traditional botnet provided by a computer device (Karim, A. *et al.*, 2014). Based on the scope of this study, Android botnet is further elaborated in detail.

2.7 Overview of Android Botnet

An Android (mobile) botnet is a network of infected Android devices controlled by a bot-master (attacker) through a command and control (C&C) server. It can cause security damage to the Android devices which include data stealing, part of the connected devices are using for their personal processing by force without knowing to the owner (Khattak *et al.*, 2014b). Distribution of spam e-mails, stealing bank credentials and identities for attacking financial services, using Distributed Denial of Service attacks for extortion, gaining criminal profits through simulating false response to advertising, infecting smartphones via websites and other similar activities are the major criminal activities of Android botnets (Tiirmaa *et al.*, 2013).

Aforesaid that smartphones have become the necessary tool in our lives to communicate with each other's. These smartphones can be used for play games, read the news, contact with others, and check the weather, online banking, maps and navigators and much more but Android applications should be installed on it (Babu *et al.*, 2015). These applications are available from third-parties sources (Amazon, 2016; PlayStore, 2017). An application calls upon any of the mobile device core functionality, like making calls, using the camera, sending text messages or picture messages, or accessing personal data storage, it allows the developer to develop richer applications (Moonsamy *et al.*, 2014). The developer can also access address book, SMS content, GPS location data, movement data by G-sensor and accelerometer, and even the information in another application (Teufl *et al.*, 2013). It is hard to differentiate between the third-party and the smartphone's core applications for Android, because they can all be built to have equal access to a smartphone's capabilities (Sears, 2014).

2.7.1 Components of Android Botnet

A typical Android botnet consists of four basic components including botmaster, command and control server, bots, and communication channel as shown in Figure 2.10. Botmaster is the entity that control botnet from the remote area while making sure any error is fixed, and that the bot does not break any of the rules of the channel or server that is logged into (Silva *et al.*, 2013). The botmaster hides their identity via proxies, The Onion Ring (TOR) and/or shells to disguise their IP Address from detection of investigators and law enforcement (Kadir *et al.*, 2015).

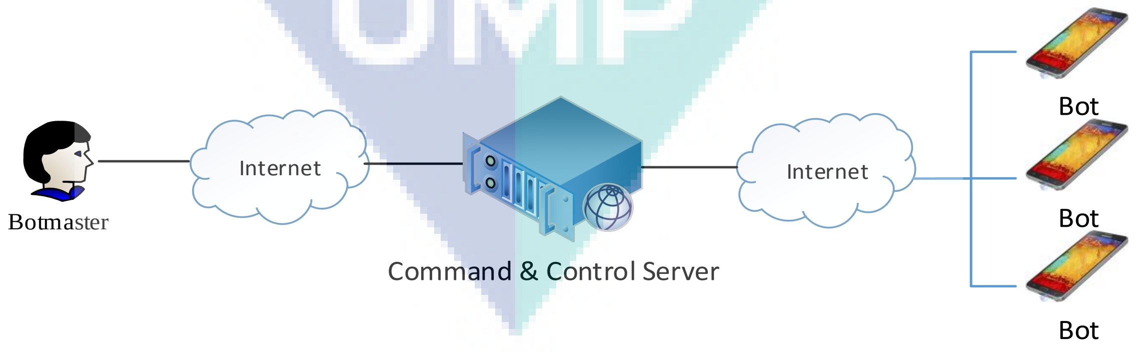


Figure 2.10 Android Botnet Components

Command and Control server is the heart of each botnet; these servers execute those commands received from the botmaster and process it according to the botmaster instruction. Communication channel allows a bot entity to take new instructions and malicious capabilities, as command by a remote individual (botmaster). These channels are used to control botnets depending on their topology. A botnet may has different C&C server topologies, like Star, Multi-Server, Hierarchical, and Random topology (Stone-Gross *et al.*, 2011).

A bot is a malicious Android application that is installed in a susceptible host through various ways which can perform a series of different harmful actions to the end user according to the botmaster commands (Karim, Ahmad *et al.*, 2014; Van Der Wagen & Pieters, 2015). Once an end user device is infected with malicious software, it receives commands and controls from the botmaster through command and control server using communication channels. However, the bots can be servant and client at the same time. Botnet communication channel refers to the protocol used by bots and botmasters to communicate with each other. Bluetooth, Internet Relay Chat (IRC), Hyper Text Transfer Protocol (HTTP)/HTTPS, peer to peer (P2P) and Voice Over Internet Protocol (VoIP) servers are used to pass information between bots and botmaster (Farina *et al.*, 2016; Silva *et al.*, 2013). In Lu & Ghorbani, (2008), botmaster is stated to create an IRC channels on the C&C server, then the compromised machines will wait for commands to perform malicious activities.

2.7.2 Android Botnet Life Cycle

The Android botnets are expected to behave like a PC botnet according to general model namely initial infection, secondary injection, connection, command and control, and maintenance (Silva *et al.*, 2013). In the initial infection stage, the botmaster exploits injected malicious code or just edit an existing one out of numerous vastly constructed Android bots over the Internet (Karim, Ahmad *et al.*, 2014). Once the bot has successfully infected a victim smartphone, it informs the C&C server and get updated timely by new commands received from botmaster. Moreover, the victim smartphone grants extra functionalities to the botmaster. Furthermore, the bot client then goes over the Internet endeavouring to grow itself to other victim smartphones (Sanz *et al.*, 2013).

In the secondary injection stage, the botmaster executes extra program on the newly acquired access which then fetch the malicious smartphone from a known location. As soon as the binary has been installed to the victim smartphone, it executes the malicious code and becomes a bot. In the connection stage, the infected smartphone attempt to initiate a connection to the C&C server through a variety of communication channels. Once this connection has been established, it joins the botnet properly. However, the maintenance stage is the last and most important stage in the botnet lifecycle, victim smartphones (bots) are commanded to update their binaries, typically to defend against new attacks or to improve their functionality. Every botnet after construction follows three types of architectures which are centralized, decentralized and hybrid (Rodríguez-Gómez *et al.*, 2013).

In a centralized botnet architecture, all the bots relate to a central command and control server to establish a communication channel with a pivotal point as shown in Figure 2.11 (A). In this architecture, the botmaster controls and supervises all bots in a botnet from a single C&C server (Stuvert & Soniya, 2015). Botmaster can communicate with the bots continuously by sending the instruction to them through these central servers. As all bots receive commands and reports to a C&C server, it is easy for botmasters to manage botnets using centralized architecture (Birundha *et al.*, 2015). In addition, a centralized botnet architecture uses two types of topologies: star topology and hierarchical topology and two types of protocols which are IRC, HTTP and HTTP Secure (HTTPS) (Khattak *et al.*, 2014a; Li, C. *et al.*, 2009). The benefits of centralized botnet architecture are as follow: low reaction time, easy way of communication, and direct feedback (Plohmann *et al.*, 2011). Furthermore, the design of centralized architecture is less complex as compared to other architectures. Moreover, the message latency and survivability are the big issue with it.

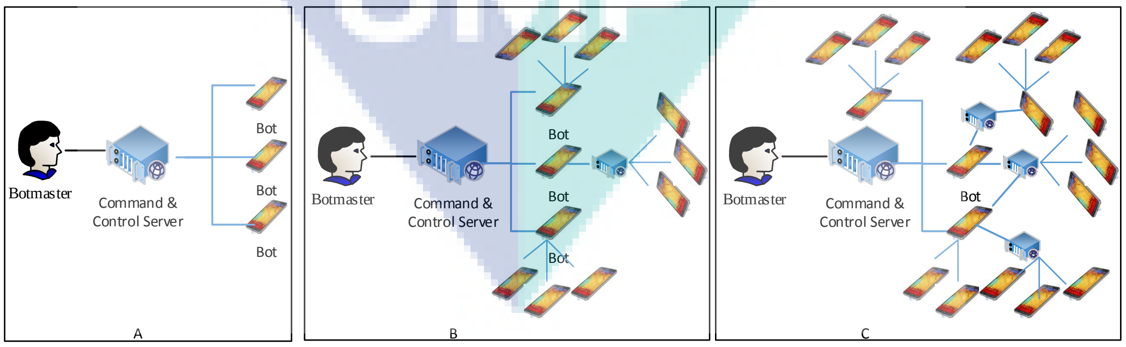


Figure 2.11 Android Botnet Architectures (A: Centralized, B: Decentralized, C: Hybrid)

Even with the benefits enumerated above, centralized botnet architecture has limitations. The main disadvantage of the centralized architecture is its maximum failure chances compared with other architectures. If the C&C server fails in a botnet having thousands of nodes, this may stop the whole botnet in very short delay (Rahman & Saudi, 2015). A centralized command and control servers make the detection of a botmaster easier as compared to decentralized and hybrid architectures (Bailey *et al.*, 2009; Zang *et al.*, 2011). Thus, it compels the botmaster to move their attention to decentralized and hybrid botnets architectures.

The Android botnets with decentralized architecture are known as a decentralized Android botnet. These botnets are more difficult to detect as compared to the centralized botnets (Tyagi & Aghila, 2011). Figure 2.11(B) described the structure of decentralized botnet architecture. It illustrates that there is no specific C&C server exists in this architecture, and all bots act like a server and client at the same time (Dong et al., 2008). The decentralized architecture is based on P2P protocols. When compare to centralized botnet architecture, the design of P2P architecture is more complex and its detection is more difficult than other botnets (Wang, Z. *et al.*, 2014). Cooke *et al.* (2005) stated that the higher survivability rate and the failure chances of the decentralized botnet over the centralized botnet are the advantages. Furthermore, hybrid architecture is the combination of centralized and decentralized architectures as illustrated in Figure 2.11(C). The hybrid architecture comprises two types of bots, namely, the servant and the client. Bots are connected to the hybrid botnet as a client or servant. Monitoring and detection of botnets with the hybrid architecture are more difficult than centralized and decentralized botnet architectures, while the design of the hybrid architecture is less complex (Wang, Ping *et al.*, 2010).

2.7.3 Android Botnet Timeline

A number of mobile botnets have evolved to degrade the performance of smartphones, for example ZeuS is a botnet specially designed for Android smartphone (Karim, Ahmad *et al.*, 2015). However, the first mobile botnet iKee.B was appeared in 2009 using SMS as C&C server while the platform was iOS (Fu *et al.*, 2015). However, this botnet does not have any criminal activities. Table 2.3 shows the mobile botnets timeline with respect to their year of creation, Name, Command & Control type, platform, cyber-criminal activities, and those permissions which are required for Android botnets. Command and Control server performs an important role in both types of botnets (traditional and mobile botnets).

Table 2.3 Android Botnet Timeline

Reference	**Detection Year**	**Name**	**C&C Type**	**Platform**	**Botnet Instructions**	**Category**	**Criminal Activities by default**	**Requires Permissions**
(Peng *et al.*, 2014)	2009	iKee.B	SMS	iOS	N/A	N/A	None	
(Nigam, 2015)	2010	SMSHowU.A	SMS	Android	Leak location; GPS; and maps through SMS	Potential Unwanted Programs (PUP)	None	N/A
(Nigam, 2015)		Zitmo.A	SMS	Symbian	ON; OFF; ADD or Set or Rem Sender; etc	Trojan	Sends SMS to Premium Phone Numbers; transferring incoming SMS to C&C server; Update C&C server and target new victims through SMS.	N/A
(Kazdagli *et al.*, 2016)	2011	Geinimi.A	HTTP	Android	ON; OFF; ADD or Set or Rem Sender;	Trojan	IMEI; IMSI;SIM; SIM State; Build Info; GPS; Board; Brand; CPU type; User; Software Version; SIM Country; SIM Operator	N/A

Table 2.3 Continued

Reference	**Detection Year**	**Name**	**C&C Type**	**Platform**	**Botnet Instructions**	**Category**	**Criminal Activities by default**	**Requires Permissions**
(Nigam, 2015)		Zitmo.A	SMS	BlackBerry	ON; OFF; ADD or Set or Rem Sender;	Trojan	Sends SMS to Premium Phone Numbers; transferring incoming SMS to C&C server; Update C&C server and target new victims through SMS.	N/A
		Zitmo.B	SMS	Symbian	ON; OFF; ADD or Set or Rem Sender;	Trojan	Sends SMS to Premium Phone Numbers; transferring incoming SMS to C&C server; Update C&C server and target new victims through SMS.	N/A
(Nigam, 2015)		Zitmo.C	SMS	Windows	ON; OFF; ADD or Set or Rem Sender;	Trojan	Sends SMS to Premium Phone Numbers; transferring incoming SMS to C&C server; Update C&C server and target new victims through SMS.	N/A

Table 2.3 Continued

Reference	Detection Year	Name	C&C Type	Platform	Botnet Instructions	Category	Criminal Activities by default	Requires Permissions
(Karim, A. *et al.*, 2014)		DroidKungFu.A	HTTP	Android	Leak location; GPS; and maps through SMS	Trojan	Send Sensitive Data; execDelete; Exploit known vulnerabilities to gain root; Install APK; execOpenUrl; execStartApp	N/A
(Nigam, 2015)	2012	Fjcon.A	HTT Phone	Android	ICCID;	Malware	Financial; Propagation of malware.	N/A
(Pieterse & Olivier, 2012)		Rootsmart	HTTP	Android	action.host start; action.boot; action. shutdown; action.install; action.installed; action.check live; action.download apk	Malware	IMEI; IMSI; cell ID; location area code; mobile network code	N/A
(Ibrahim & Hatim, 2012)		TigerBot.A	SMS	Android	Change APN; Notify of SIM change; Kill running process	Trojan	IMEI	N/A
(Wang, P. *et al.*, 2015)	2013	Stealer.B	HTTP and SMS	Android	HTTP: time; sms; send; delete; smscf SMS: ServerKey +001; +002; anything	Malware	IMEI; IMSI; Contacts	READ_SMS; INTERNET; RECEIVE_BOOT_COMPLETED; READ_PHONE_STATE; RECEIVE_SMS; READ_CONTACTS; SEND_SMS; WRITE_EXTERNAL_STORAGE

Table 2.3 Continued

Reference	Detection Year	Name	C&C Type	Platform	Botnet Instructions	Category	Criminal Activities by default	Requires Permissions
(Yin, 2014)		Tascudap.A	HTTP	Android	time; sms; send; delete;smscf SMS: ServerKey + 001; 002; anything	Malware	Specify time when trojan should next contact C&C; send SMS; delete SMS from phone; selective SMS hiding; start application; forward received SMS; update	READ_SMS; ACCESS_NETWORK: INTERNET; READ_PHONE_STATE; RECEIVE_SMS; READ_CONTACTS; SEND_SMS; WRITE_EXTERNAL_STORAGA;
(Nigam, 2015)		BadNews.A	HTTP	Android	news; showpage; install; showinstall; iconpage; coninstall; newdomen; seconddomen; stop; testpost	Trojan	Propagation of possible malware; download and installation of APK	RECEIVE_BOOT_COMPLETED; SEND_SMS; RECEIVE_SMS; INTERNET; ACCEESS_INTERNAL_MEMORY; ACCESS_EXTENAL_MEMORY;
(Nigam, 2015)		Spamsold.A	SMS	Android	Display same icon on the menu; remain the image same; while the name may change; install APK once click.	Trojan	Sends SMS spam messages without the user's consent	INTERNET; CHANGE_COMPONENT_ENABLED; RECEIVE_SMS; READ_SMS; SEND_SMS; WRITE_SMS; RECEIVE_SMS; RAISED_THREAD_PRIORITY; READ_CONTACTS; WRITE_EXTERNAL; RECEIVE_BOOT_COMPLETED; WAKE_LOCK;

Table 2.3 Continued

Reference	Detection Year	Name	C&C Type	Platform	Botnet Instructions	Category	Criminal Activities by default	Requires Permissions
(Nigam, 2015)	2014	FrictSpy.E3	HTTP; SMS	Android	Command and Control to execute Malware activities; such as calls record; use camera for pictures and videos; use mic for recording voice.	PUP	Incoming/Outgoing call; Incoming/Outgoing SMS; GPS location information; URLs that the device user accesses	ACCESS_NETWORK_STATE; CALL_PHONE; GET_TASKS; INTERNET; READ_PHONE_STATE; READ_SMS; RECEIVE_BOOT_COMPLETED; RECEIVE_SMS; SEND_SMS; SYSTEM_ALERT_WINDOW; WAKE_LOCK; WRITE_SMS;
(Nigam, 2015)		Geinimi.A	HTTP	Android	ON; OFF; ADD or Set or Rem Sender;	Trojan	User; Software Version; IMEI; SIM State; CPU type; SIM Country; IMSI;SIM; SIM Operator Build Info; GPS; Board; Brand;	CALL_PHONE; GET_TASKS; INTERNET; READ_PHONE_STATE; READ_SMS; RECEIVE_BOOT_COMPLETED; RECEIVE_SMS; SEND_SMS; SYSTEM_ALERT_WINDOW; WAKE_LOCK; WRITE_SMS;
(Nigam, 2015)		SpyBubb.A	SMS	Android	Leak location; GPS; and maps through SMS; HTTP: time; sms; send; delete; smscf SMS: ServerKey +001; +002; anything	PUP	Collect SMS; call; Fine Location; Coarse Location; GPS; Device Infor like IMEI; IMSI etc Share Phone information to vendor site.	N/A

Table 2.3 Continued

Reference	Detection Year	Name	C&C Type	Platform	Botnet Instructions	Category	Criminal Activities by default	Requires Permissions
(Nigam, 2015)	2015	Leech.A	HTTP	Android	action.host start; action.boot; action. shutdown; action.install; action.installed; action.check live; action.download apk	Malware	Install itself persistently; run with full privileges; unwanted payment through SMS; Spying activities; Dynamically load command and control server.	ACCESS_NETWORK_STATE; ACCESS_WIFI_STATE; READ_PHONE_STATE; INTERNET; WAKE_LOCK;
(Nigam, 2015)		Tediss	SMS	Android	N/A	Malware	Monitor Calls; SMS; and Conversation Applications.	CALL_PHONE; GET_TASKS; INTERNET; READ_PHONE_STATE; READ_SMS;RECEIVE_BOOT_COMPLETED; RECEIVE_SMS; SEND_SMS; SYSTEM_ALERT_WINDOW; WAKE_LOCK; WRITE_SMS;
(Nigam, 2015)		WormHole.A	HTTP and SMS	Android	action.host start; action.boot; action. shutdown; action.install; action.installed; action.check live; action.download apk	PUP	Install applications without notification; location information; add contact items; monitor list of applications	READ_EXTERNAL_STORAGE; READ_PHONE_STATE; READ_NETWORK_STATE; INTERNET; READ_INTERNAL_STORAGE; WAKE_LOCK; READ_COARS_LOCATION;

Table 2.3 Continued

Reference	Detection Year	Name	C&C Type	Platform	Botnet Instructions	Category	Criminal Activities by default	Requires Permissions
(Nigam, 2015)		SilverPush.A	HTTP and SMS	Android	HTTP: time; sms; send; delete; smscf SMS: ServerKey +001; +002; anything; ON; OFF; ADD or Set or Rem Sender;	PUP	IMEI number; Operating system version; Location; Potentially the identity of the owner; Behavior of users using TVs; web browsers; and Radios.	ACCESS_NETWORK_STATE; CALL_PHONE; GET_TASKS; INTERNET; READ_PHONE_STATE; READ_SMS; RECEIVE_SMS; SEND_SMS; WRITE_SMS;
(Fan *et al.*, 2017)	2016	MazarBOT.A	SMS	Android	N/A	Malware	Sends premium SMS; exfiltrate sensitive information; and steal the received SMS messages; by setting up a backdoor on device.	ACCESS_NETWORK_STATE; CALL_PHONE; GET_TASKS; INTERNET; READ_PHONE_STATE; READ_SMS; RECEIVE_BOOT_COMPLETED; RECEIVE_SMS; SEND_SMS; SYSTEM_ALERT_WINDOW; WAKE_LOCK; WRITE_SMS;

Table 2.3 Continued

Reference	Detection Year	Name	C&C Type	Platform	Botnet Instructions	Category	Criminal Activities by default	Requires Permissions
(Fan *et al.*, 2017)		Morder.A	HTTP and SMS	Android	Command and Control to execute Malware activities; such as calls record; use camera for pictures and videos; use mic for recording voice.	Trojan	Track location; Leak contacts to C&C Upload data from SD Card to C&C; Delete or download files in the infected device; Leak phone call history; Take pictures with the camera; Record audio and calls; Execute shell commands	ACCESS_NETWORK_STATE; CALL_PHONE; GET_TASKS; ACCESS_FINE_LOCATION; ACCESS_COARS_LOCATION; INTERNET; READ_PHONE_STATE; READ_SMS; RECEIVE_BOOT_COMPLETED; RECEIVE_SMS; SEND_SMS; SYSTEM_ALERT_WINDOW; WAKE_LOCK; WRITE_SMS;
(Fan *et al.*, 2017)		Smishing.D	SMS	Android	time; sms; send; delete; smscf SMS: ServerKey +001; +002; anything; ON; OFF; ADD or Set or Rem Sender;	Malware Phishing	Detect Text Messages; access fraudulent fake bank URL; steal user's sensitive credential; password stealing; additional information stealing.	ACCESS_NETWORK_STATE; CALL_PHONE; GET_TASKS; INTERNET; READ_PHONE_STATE; READ_SMS; RECEIVE_SMS; SEND_SMS; WRITE_SMS;

NA=Not Available, SMS= Short Messaging Services, MMS= Multimedia Messaging Services, HTTP= Hyper-text Transfer Protocol, ICMP= Internet Control Message Protocol, SMTP= Simple Mail Transfer Protocol, UDP= User Datagram Protocol, FTP= File Transfer Protocol, IRC= Internet Relay Chat, TCP= Transmission Control Protocol, P2P= Peer-to-Peer,

2.8 Botnet Detection Approach

In this section, the most significant advances in botnet detection approaches for smartphones are described. There are different botnet detection approaches have been proposed. These approaches aim to identify where and how botnet manifests by constantly monitoring various smartphone-based features. Basically, the existing detection approaches are divided into three main categories based on their types, namely static, dynamic, and hybrid as demonstrated in Figure 2.12 (Silva *et al.*, 2013).

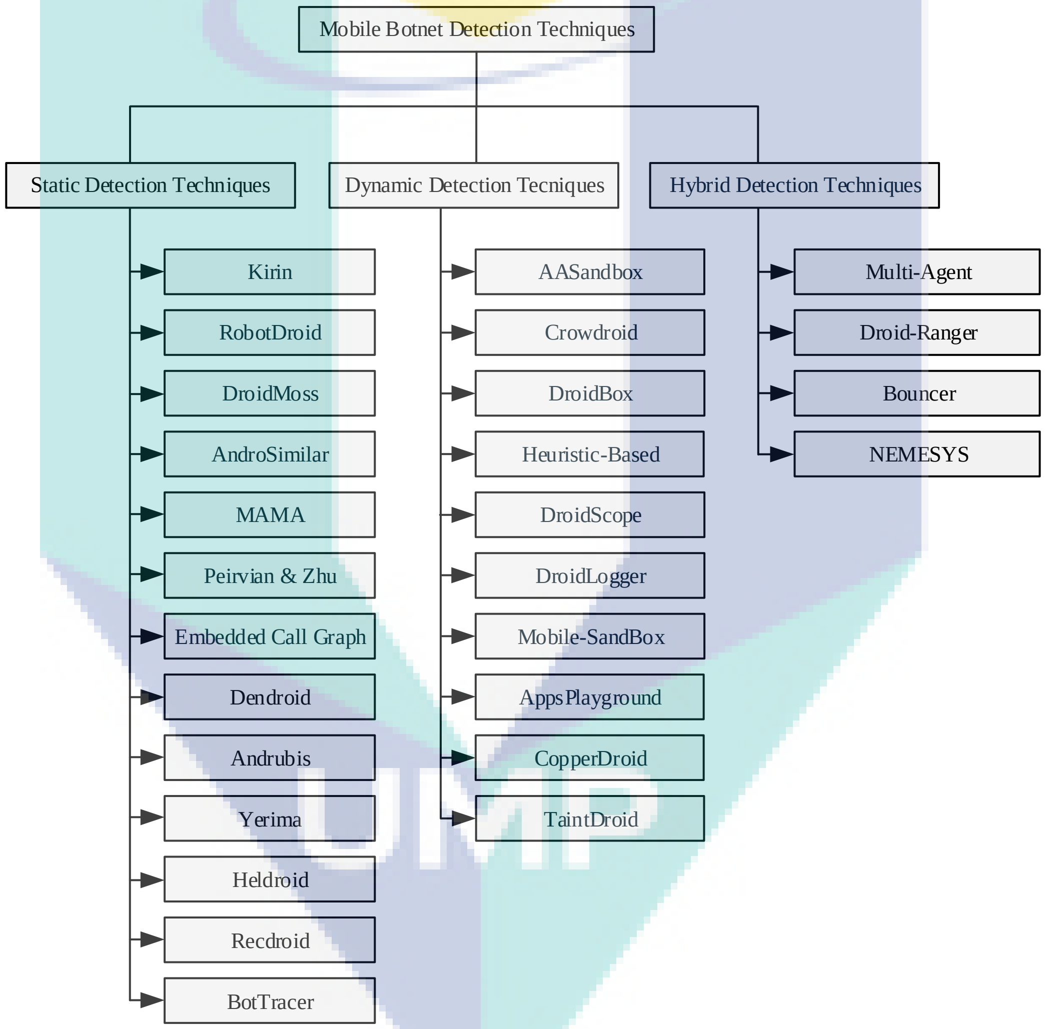


Figure 2.12 Mobile Botnet Detection Approaches

The process of unpacking the APK (Android installation files) for examining the static features such as permissions, activities, broadcast receivers, and services to detect botware applications is known as Static analysis. Static detection approaches are well known in traditional botnet detection and have recently gained popularity as an efficient mechanism for Android botnet detection. This approach is relatively fast and has been widely used in the preliminary analysis to search for suspicious strings or blocks of code.

However, dynamic analysis (also known as behavioural-based analysis) seeks to identify malicious behaviours namely system calls, files, network access and memory modifications. It can be performed after deploying and executing the Android applications on an emulator. These approaches require some human or automated interaction with the app, as malicious behaviour is sometimes triggered only after certain events occur. Moreover, it seems hard to apply dynamic approach in mobile environment due to limited resources such as CPU, power, and memory of the smartphones (Lindorfer *et al.*, 2015). Furthermore, Hybrid analysis is the optimum approach that seeks to identify malicious applications by utilizing the static and dynamic analyses. However, this study only focuses on the static analysis approaches.

2.8.1 Static Detection Techniques

Static detection techniques are well known in traditional botnet detection and have recently gained popularity as an efficient mechanism for market protection. This technique attempts to identify malicious code by unpacking and disassembling the Android applications.

Enck, W. *et al.* (2009) proposed Kirin security service which is an OS-level protection that provides enhanced security mechanisms for Android smartphone applications. This approach performs lightweight certification of applications to mitigate malware at installation time with modification of Android applications installer (Jang, Kang, *et al.*, 2016). Kirin has different parts of their security rules, and a well-known combination of permissions is the most important part in these rules (Wang, Z. *et al.*, 2016). In order to define these security rules, a detailed understanding of malware and protection techniques are required, which are usually performed by security experts (Zonouz *et al.*, 2013). Furthermore, access to the sensitive information is prevented and once an information enters the application, no additional mediation occurs (Ramaki *et al.*, 2015).

Zhao, M. *et al.* (2012) proposed RobotDroid; an Android malware detection technique that is based on SVM machine learning classifier algorithm. This technique was focused on the signatures of the applications. RobotDroid has the capabilities to detect unknown malware such as Plankton, Gemini and DroidDream (Jung et al., 2015). However, the key limitation of the RobotDroid framework is that it can only be used for a few types of malware (Fereidooni *et al.*, 2016).

Zhou and Jiang (2012) proposed DroidMoss which is based on fuzzy hashing technique that effectively localize and detect the repackaged and injected applications. This technique uses repackaging technique to detect injected malicious codes in the Android applications of the existing mobile applications market (Song *et al.*, 2016). The main feature of the applications used in this technique is Dalvik bytecodes that is made up of operands and opcodes (Rastogi, S. *et al.*, 2016). DroidMoss calculates fuzzy hashes on each N sequential opcodes and then apply a measure function on each two applications to realize their similarity quantitatively. The usage of DroidMoss is limited to identifying repackaged official Android market applications. The main limitation of this approach is the consideration of DEX bytecode only, and opcode sequence that do not contain important information and hereby, generate false negatives (Faruki *et al.*, 2015). Furthermore, this system is also low robust, and detection may fail if big chunks of code have been added to the original application (Gurulian *et al.*, 2016).

Faruki *et al.* (2013) proposed AndroSimilar that detects Android malware regions of statistical similarity starting from the .dex file. This method employs the similarity digest hashing system on byte stream based on robust statistical malicious features (Faruki *et al.*, 2015). Similarly, a digest hashing scheme uses this feature to generate a list of signatures for this app. Here, the feature values between 100 and 990 are selected and the rest are discarded using Bloom filter (Canfora *et al.*, 2016). A set of malicious signatures are generated and thus a database of signatures is created. For testing a sample app, its signature is created in the same way as above which is matched against the signature database and is considered as malware if the similarity score crosses 35 % (Sharma *et al.*, 2016). Authors obtain an accuracy of 72.27% using a dataset of 101 malicious applications. Androsimilar performs at file level as an alternative to code in decompiling; therefore, control of shared library is not protected. Also, porting the approach to constrained memory and the strong database remains a concern still (Alam *et al.*, 2017).

A study (Sanz *et al.*, 2013) MAMA to discover malicious applications by using the difference in Android application permissions that the application request upon installation. Android permissions are coarse-grained (Sokolova *et al.*, 2017). For example, the INTERNET permission does not have the capability to restrict access to a particular Uniform Resource Locator (URL). READ_PHONE_STATE allows an app to identify whether the device rings or is on hold. At the same time, it also allows the app to read the sensitive information such as device identifiers. Permissions such as WRITE_SETTINGS, CAMERA are broadly defined, thus it violates the least privilege access principle. Access to WRITE_CONTACTS or WRITE_SMS does not imply the access to READ_CONTACTS or READ_SMS permissions (Google.com, 2016). Thus, permissions are not hierarchical, and they must be separately requested by the developer. At the install time, the user is forced to grant either all permissions or deny the app installation. Hence the dangerous permissions cannot be avoided at the install time. Moreover, the users cannot differentiate between the necessity and its imperative misuse which may expose for exploitation.

In this approach, the authors extracted static features from the androidmanifest.xml file of 666 Android applications. They have used machine learning techniques such as K-Nearest Neighbors (K-NN), Decision Trees, Bayesian networks, Support Vector Machines (SVM) to detect malicious applications. In this study, K-NN is used due to its simplicity in classifying the instance into different classes; a decision tree is used because of its combination and easy implementation (Narang et al., 2016). Bayesian network is employed for determining the probability of a hypothesis certainty. On the long run, SVM is used to overcome the problem of kernel functions which may lead the technique to the non-linear classification surface (Wang, W. et al., 2017).

Likewise, WEKA tool was adopted for the evaluation of machine learning algorithms (Hall et al., 2009). The dataset was divided using the k-fold cross validation technique. The k-fold means to divide the input datasets in 'k times' in 'k numbers' of subsets using one shaping sample data set, known as a test set. In their study, they used 130 numbers of permissions and other features separately as an input. Moreover, they successful obtained satisfactory results with 87.41% accuracy for permission features and 86.09% for the API calls. However, with the combination of permission features that was used they got 94.83% which sounds good in terms of malware detection. Remarkably, this technique cannot detect mobile botnets, because of their specific and unique features.

In other words, study of (Peiravian & Zhu, 2013) developed a machine learning framework to analyze benign and malicious applications by using the permissions and API calls as features input. They have extracted these requested permissions from the

Androidmanifest.xml file and API calls from .DEX classes of each Android applications (Desnos, 2011a). In this study, authors examined 2400 Android malware and benign applications in total. The authors achieved 96.88% of accuracy by selecting a high number of features. This shows that the accuracy can be improved when a greater number of features is selected from both malware and benign applications.

In particular, this study has some limitations about complexity, by extracting and selecting a greater number of features. With the selection of more number of features, the accuracy is improved while the memory and time complexity are increased as well (Sokolova et al., 2017). This framework specifically focused on the malware applications. Using these approaches mobile botnets cannot be detected in Android devices. This study has almost the same nature with aforementioned study on MAMA in Section 2.7.1 (Sanz et al., 2013) by using the same features. They achieved 94.83% accuracy when 130 features were examined.

Gascon *et al.* (2013) proposed Embedded call graphs technique based on static approach using SVM classification algorithm. The use of call graph kernel for malware detection allows for extraction of the code into a readable file that make the structure learning possible (Li, L. et al., 2017). This technique can be used to find similarities between android applications samples. The key concept of this technique is functioned call graphs, while obfuscation resistance is the major contribution (Gascon et al., 2013). It specially observes the assembly level analysis and support vector machine implementation. The main disadvantage of this technique is inability to decide the static call graph construction, while the time and space complexity are high and large (Sharma et al., 2016).

Suarez Tangil *et al.* (2014) proposed Dendroid approach based on text mining and information retrieval techniques. In this technique, the Code Chunks (CC) are extracted for further analysing and classified the code structures in malware families (Faruki et al., 2015). The authors present a simple way to measure the similarity among malicious applications by formulating the modelling process (Suarez Tangil et al., 2014). The experiment performed over 33 families had 1249 malware applications (Sharma et al., 2016). This approach provides the automatic classification of zero-day malware samples, which is based on applications code structure. According to time and accuracy, this technique is very fast and accurate, while having a high scalability (Skovoroda & Gamayunov, 2015). However this technique has some limitation in terms of feature vector growth, new families creating issues, and the strategies of obfuscation are not implemented (Tam et al., 2017).

Lindorfer *et al.* (2014) introduced Andrubis which is a cloud-based malware detection technique. This technique combines both static and dynamic analysis on Dalvik VM and System level (Jang, Kang, *et al.*, 2016). First, it performs the static analysis by extracting the information including broadcast receivers, requested permissions, activities, services, SDK version, package name, from the application manifest and its bytecode. Andrubis uses the modified DroidBox output to generate XML files that contains the analysis results (Abdullah J. Alzahrani, 2014). While in the dynamic stage, it executes the application in a complete Android environment, during the execution its action is monitored at both the Dalvik and the system level (Faruki et al., 2015). Other than this, Andrubis provides a web interface for users to submit Android applications and has collected a dataset of over one million Android applications including 40% malware. The only disadvantage of this technique is that it cannot track native code (Rasthofer *et al.*, 2015; Xu, L. *et al.*, 2016). API calls that are frequently happening in the botnet are extracted from the Dalvik code, while the Andrubis is limited to the applications API level 8.

Yerima *et al.* (2014a) developed a proactive machine learning approach that is based on Bayesian classification and aimed to detect zero-day Android malware attacks using static analysis approach. This approach has three main components which are decompression, identification, and classification. First, an application is decompressed by reverse engineering to extract features from AndroidManifest.xml and .DEX classes by using Dalvik VM (Feizollah et al., 2017). All the extracted features are stored in a file with the .csv extension for further analysis. In the identification step, this component converts the extracted features file to a readable form for further analysis. While in the classification process the Bayesian algorithm classifies the malware and benign applications as a result (Hall et al., 2009). This approach is based on large existing malware set of 49 families. Specifically, this technique achieved approximately 92.1% accuracy by using a set of 30 static features.

This research work focusses on the Android malware identification only. By using this approach, a mobile botnet cannot be detected. They used a very few number of features as compared to the aforementioned study (Peiravian & Zhu, 2013) which affect the accuracy of malware detection. According to this study, the time taken for features extraction and computation is decreased to a tune of 77%. However, in another study, it is shown that by using these features the time taken for features extraction is increased with an amount of 28% (Karim, A. et al., 2015). In this approach accuracy and time consumption are the key issues that need to be addressed in future.

Heldroid is device based ransomware detection technique that is based on the building blocks (Andronio et al., 2015). Both static and dynamic approaches are used to analyse Android applications. In addition, a light-weight emulation is used to find the flows of function calls. The technique was tested with 187326 samples and produced 99% correctly identification of ransomware (Li, L. et al., 2017). They obtained their best results with respect to ransomware only. It sounds good for the ransomware detection in mobile devices, on the other hand, it cannot detect botnet and other malware which is the main limitation of this technique (Sadeghi et al., 2017). Space and time complexity also exist in this technique because of its devices-based approach.

Liu *et al.* (2008) introduced RecDroid; a clustering-based method to detect bot users controlled by the same masters. RecDroid is an Android permissions recommendation framework which allows users to grant application permissions requests in a fine-grained manner (Rashidi & Fung, 2016). The key idea behind the RecDroid is to collect the expert users' responses to a permission request and recommend them to inexperienced users (Rashidi et al., 2017). It followed two steps with the first approach involves analysis the common features of bot users and constructed a graph based on their similarity (Kirubavathi & Anitha, 2017). Then, a hierarchical clustering method is employed to group those users together based on their distance which is defined using similarity. This approach is limited to detect only simulated bot user profiles (Rashidi et al., 2016).

BotTracer is proposed by (Rashidi & Fung, 2016) and is a clustering based method to detect bot users controlled by the same masters. Their main part of the proposed method is to plot the Android users in various groups on the basis of their similarity. It is an Android permissions recommendation framework which allows users to grant application permissions requests in a fine-grained manner. In Android applications, permissions are the main factor during installation that cannot be ignored. For example, the INTERNET permission does not have the capability to restrict access to a particular Uniform Resource Locator (URL), READ_PHONE_STATE allows an app to identify whether the device rings or is on hold (Felt et al., 2012).

The key idea behind this technique is to collect the expert users' responses to a permission request and recommend them to inexperienced users. BotTracer is in two phases with the first step on analysis of the common features of bot users for the construction of a graph that is based on their similarity. Then, a hierarchical clustering method is employed to group those users together based on their distance which is defined using similarity. This approach has many limitations in terms of botnet detection. First, it detects the simulated bot user profiles only. Secondly, it considers the DEX bytecode only, while it ignores the native code and app resources. Thirdly, the opcode sequence does not include high-level semantic information and hence generates false negatives. With these limitations, smart adversary can easily bypass this technique using code transformation techniques such as inserting junk bytecode, restructure methods, and alter control flow to evade the BotTracer prototype.

The complete list of mobile botnet detection techniques using static approach are summarized in Table 2.4. The technique column represents the approach used, followed by the year column. Furthermore, the key concept presents the rules on which these approaches are detecting botware applications namely permission based, behavior based, and signature based. Major contribution shows detection approaches strength, while limitations shows the disadvantages of each approach.

Table 2.4 Mobile Botnet Detection Techniques using Static Approach

Refernces	**Techniques**	**Year**	**Key Concept**			**Major Contribution**	**Observations**	**Limitation**
			Permission Based	**Behaviour Based**	**Signature Based**			
(Enck, W. *et al.*, 2009)	Kirin	2009	✓	✗	✗	Used rules to detect malware in install time.	It is logic based tool, ensure permission needed by application are met by global safety invariants.	No access to sensitive information of application. Cannot detect new malware (Enck, W. *et al.*, 2014).
(Zhao, M. *et al.*, 2012)	Robotdroid	2012	✓	✓	✓	Detection of unknown malware such as Plankton, DroidDream, and Gemini.	Signature recognition	It is limited to detect some specific types of malware families such as Plankton, DroidDream, and Gemini (Narudin *et al.*, 2016).
(Zhou *et al.*, 2012)	DroidMOSS	2012	✓	✗	✗	Fuzzy Hashing Technique,	measure the similarity between two are more different applications	It is limited to identifying repackaged official Android market applications (Enck, W. *et al.*, 2014).
(Faruki *et al.*, 2013)	AndroSimilar	2013	✓	✓	✓	Improbable signature generation, thwart obfuscation and repackaging	Entropy, signatures, fuzzy hashing	Limited malware dataset and it can detect only simulated bot users, more false positives and poor detection rate. Unable to detect new malware (Alam *et al.*, 2017).
(Sanz et al., 2013)	MAMA	2013	✓	✗	✗	Machine Learning classifiers based on permissions and the features from the manifest file.	Over 2000 applications are analyzed for permissions. Majority of the malicious applications requested network connectivity.	Extraction of more number of sub features making it high power and space consumption, draining the battery. It do not prevent installation of malware (Narudin *et al.*, 2016).
(Peiravian & Zhu, 2013)	--	2013	✓	✗	✗	Permissions, API calls and the combination of both are used to detect malicious applications	2400 real world applications are used to validate the performance of algorithm.	Facing issue in new family creation of botware also cannot prevent installation of botware (Tchakounté & Hayata, 2016).

Table 2.4 Continued

Refernces	Techniques	Year	Key Concept: Permission Based	Key Concept: Behaviour Based	Key Concept: Signature Based	Major Contribution	Observations	Limitation
(Gascon *et al.*, 2013)	Embedded call graph	2013	✓	✓	×	Obfuscation resistance	Assembly level analysis, SVM implement	Undecidability of static call graph construction. Cannot detect new malware .
(Suarez-Tangil *et al.*, 2014)	Dendroid	2014	✓	✓	✓	Unknown malware classification, fast and scalable, dendograms	Malware signatures, VSM, 1-NN, Malware evaluation	No obfuscation resist, large feature vectors. Cannot detect new malware (Suarez Tangil *et al.*, 2014).
(Lindorfer *et al.*, 2014)	ANDRUBIS	2014	×	✓	✓	Static Analysis on both Dalvik VM and System Level	Fully automated technique, perform dynamic, static, and auxiliary analysis	Analysis consume more space, cannot be used for latest Android applications (Karim, Salleh, Khan, *et al.*, 2016).
(Yerima *et al.*, 2014a)	---	2014	✓	×	×	Applied static analysis-based Bayesian classification for proactive android malware detection	Observed permissions, code-based features and mixed features	feature vector growth, it does not allow malware families classification, and the strategies of obfuscation (Canfora *et al.*, 2016)
(Andronio *et al.*, 2015)	HELDROID	2015	✓	×	×	Different APIs, specifically SMS APIs and functions to detect crypto-ransomware and locker-ransomware	Monitor different multiple source data sense,	It does not track implicit control flows due to performance overhead (Al-rimy *et al.*, 2018).
(Rashidi & Fung, 2016)	RecDroid	2016	✓	×	✓	Improbable signature generation, obfuscation & repackaging	Perform static analysis for permissions, and API_calls	This approach is limited to detect only simulated bots user profile (Kirubavathi & Anitha, 2017).
(Rashidi & Fung, 2016)	BotTracer	2016	✓	×	×	Runs a virtual machine that start automatically without interaction of human. It has the capability to detect bot when it begins a malicious activity.	It observe the three stages of a bot namely injection, update and attack.	It require high level compuations due to virtual machine degredation of host performance.It will unable to detect the moderate bots due to the capability of checking the virtual machine presence (Alauthman, 2016).

2.8.2 Dynamic Detection Techniques

Dynamic detection techniques seek to identify malicious behaviours after deploying and executing the applications on an emulator or a controlled device. These techniques require some human or automated interaction with the app, as malicious behaviour is sometimes triggered only after certain events occur. Some of the dynamic detection techniques are listed in the following paragraphs with more detail.

AASandbox was the first technique perform both static and dynamic analysis of the Android applications proposed by (Bläsing *et al.*, 2010). The static analysis scans the Android applications for malicious patterns without the installation on the Android platform (Jang, Yun, *et al.*, 2016). However, in the dynamic analysis the Android application is executed in a fully isolated platform called a sandbox (Sanz *et al.*, 2013). It also intervenes and logs low–level interaction with the system for further analysis during the application execution. In contrast, both the detection algorithm and sandbox algorithm are implemented in the cloud. AASandbox uses a system known as foot-printing approach for detecting suspicious Android applications. It logs the execution time, the system call name and the identifications of each processes (Alazab *et al.*, 2012). In early days when AASandbox was proposed, there were no known Botnet malware samples available to evaluate this technique (Suarez Tangil *et al.*, 2014). This seems to be unmaintained nowadays.

Burguera *et al.*, (2011) proposed Crowdroid, this is a dynamic approach based on the behavior of Android applications. Crowdroid is a lightweight application and is available on the Google play store for download and installation on the devices (Xu, J. *et al.*, 2013). It monitors and collects the API calls of those apps which are running on mobile devices and send them to the centralized server after pre-processing (Narudin *et al.*, 2016). With the application of cluster algorithm to evaluate these Android applications, it is able to detect self-written malware as well (Skovoroda & Gamayunov, 2015). However, Crowdroid is based on the Strace, which extract system calls from the applications after installation (Levin *et al.*, 1991). Crowdroid cannot detect malicious behavior during installation process, as it depends on the functionality of Strace (Jang, Yun, *et al.*, 2016).

DroidBox is a sandbox based applications for behavioral analysis as proposed by (Desnos & Lantz, 2011). It is basically TaintDroid with some extra Dalvik virtual machine modifications that log specific API calls (Alazab *et al.*, 2012). This technique can effectively analyze Android application by building logs of all data accessed by the application on the system (Junaid *et al.*, 2016). However, it lacks in executing applications prior to Android version 4.2 (Tam *et al.*, 2017). DroidBox is an open source package for dynamic analysis which cannot be used explicitly for large datasets because of its limited resultant parameters and deficiency to execute latest Android applications (Karim, Salleh, & Khan, 2016).

Heuristic based botnet detection monitor network traffic including IRC traffic, HTTP traffic, and unclassified application traffic to identify malware (Franklin *et al.*, 2008). A processor is configured to monitor the behavior which indicates the suspicious network traffic (Moghaddam & Abbaspour, 2014). Heuristic botnet detection technique uses new and precise traffic patterns to identify C&C activities with an improved accuracy and low false positive rate (Zhou *et al.*, 2012). This traffic pattern is used to identify botnet C&C activities, various heuristic techniques as described herein with respect to various embodiments. This technique has some drawbacks that undermines its efficiency such as length of monitoring time is not clear and the condition that trigger malicious behaviour is not evident (Shameli *et al.*, 2014). The Heuristic technique might be more expensive regarding computationally and resource consuming.

Yan & Yin. (2012) proposed DroidScope, it is a fine-grained dynamic binary instrumentation tool for Android OS that rebuilds two level of semantic information: OS and Java. It provides an instrumentation interface which can be used to write plug-ins (Jiang & Xuxian, 2013). Some of the plug-ins has already been implemented such as API tracing, native instruction tracing, Dalvik instruction tracing and taint tracking. DroidScope works entirely on the emulator level and requires no changes to the Android sources (Jang, Kang, *et al.*, 2016). It runs the analysis outside the smartphone software stack and can analyse kernel-level attacks. This system has a big drawback for not detecting real-time attacks (Enck, W. *et al.*, 2014). However, the second drawback is ignorance of covering the subtleties from the real devices (Fan *et al.*, 2017).

Another technique is DroidLogger, this is a dynamic light-weight method for understanding the behaviour of Android applications by logging applications API's and corresponding arguments (Dai, S. *et al.*, 2012). This system can capture not only the suspicious API invoked by the application but also the arguments used by the suspicious

API (Karim, A. *et al.*, 2015). There are some limitations in the static analysis which is solved in this technique by getting the plain text of that data which is encrypted. Similarly, this technique is based on the run-time information (Faruki *et al.*, 2015). The detection of modified code by a malware, the string type of the suspicious API's arguments, and that malware which are using native code cannot be detected with DroidLogger.

Mobile-SandBox is a static and dynamic analysis system at the same time, which was proposed by (Spreitzenbarth *et al.*, 2015) and was made available to the public. In this technique, the comparison of applications occur in different stages such as comparing the hash value with the VirustTotal database of the running application in the first stage, (Ghafir & Prenosil, 2016; Virustotal, 2017). In the second step, it extracts the Manifest file for permissions, background services, broadcast receivers, and intents (Google.com, 2016). This technique also extracts the API calls from the Dalvik bytecode which happen frequently in Botnets. Due to user interface, Mobile-SandBox, is very easy to submit applications for static and dynamic analysis (Sharma *et al.*, 2016). A user can easily upload an application for static and dynamic analysis to the Mobile-SandBox by using the user interface. While in some aspect, Mobile-SandBox seems to be unable to cope with their submission load.

Rastogi *et al.* (2013) proposed AppsPlayground which is based on TaintDroid. This is a scalable automatic dynamic analysis system that detects possible data leaks. It employs a Java app that connects to an emulator running on a modified version of the OS and examined the applications dynamic features (Skovoroda & Gamayunov, 2015). It determines whether the application is involved in malicious activities are being carried out by monitoring sensitive API and system calls or tracking personal data leakage (Suarez Tangil *et al.*, 2014). However, AppsPlayground has one key drawback of requiring a modified Android framework for malicious applications analysis (Jang, Kang, *et al.*, 2016).

Reina *et al.* (2013) proposed CopperDroid, a dynamic detection system for system-call centric. This system is built on top of the quick emulator (QEMU). CopperDroid has a combined analysis to recreate the dynamic features of a malware program by leveraging operating system explicit information, such as system call and Android OS credentials such as private data leakage (Jang, Kang, *et al.*, 2016; Tam *et al.*, 2017; Tam *et al.*, 2015). To the best of our knowledge, this is the first technique that performs system call monitoring of the Android applications out-of-the-box through virtual machine introspection (VMI) by reconstructing the Dalvik behavior with monitoring Binder communication (Lindorfer *et al.*, 2014). CopperDroid carried the binder analysis to perform the reconstruction of high-level Android-specific behavior. It is available publicly as a web application, in which user can submit their samples (Reina *et al.*, 2013).

TaintDroid is a system-wide dynamic taint tracking and analysis system for simultaneously tracking multiple sources of sensitive data (Enck, W. H., 2011). This technique monitors methods, variables, files, and messages during the application execution according to data flow (Suarez *et al.*, 2014). TaintDroid using tag chunk to keep track of data in order to find information leakage at runtime (Rastogi, V. *et al.*, 2013). Information flow tracking needs lots of memory (Ongtang *et al.*, 2012). However, none of these schemes is energy-efficient; hence they are not suitable for resource constrained mobile platforms.

2.8.3 Hybrid Detection Techniques

Hybrid detection techniques perform static and dynamic analysis at the same time to detect malicious applications. Some of the detection techniques that use the hybrid approach are listed in the following paragraph.

Szymczyk, (2009) proposed Multi-Agent Bot Detection System (MABDS) which was based on hybrid approach. MABDS combines multiple agents such as administrative agent, user agent, a central knowledge database, system analysis, honeypots, agent collections and network analysis (Silva *et al.*, 2013). In this technique, each agent observes traffic using different sensors by implementing the Markov chain model to perform the dynamic risk assessment (Shameli *et al.*, 2014). These systems in multifaceted, piercing, real-time domains involve autonomous agents that should act as a team to compete against malware (Castiglione *et al.*, 2014). The slow convergence of new signature with the knowledge database is the key limitation of this technique. Furthermore, the new signatures updates are another limitation of this system (Karim, A. *et al.*, 2014).

Zhou *et al.* (2012) proposed DroidRanger which is a combination of two systems based on permissions behaviour foot-printing and heuristic based filtering. This technique applies to both static and dynamic approaches to detect malicious applications in the existing Android markets (Jang, Kang, *et al.*, 2016). Permissions based behaviour foot-printing is specialized for detection known malware while the heuristic based filtering is fashioned for detecting unknown malware Android applications (Song *et al.*, 2016). Despite, the advancements in the detection approaches applied by DroidRanger, it also has some limitations. For example, it requires the manual operation for analysing and collecting behaviour of Android applications (Babu *et al.*, 2015). It was reported in a related study by (Spreitzenbarth *et al.*, 2015), that DroidRanger uses manual operation which may takes more times as compare to other detection techniques.

The Oberheide & Miller (2012) proposed Bouncer. This provides the static and dynamic scanning together with Android applications which is performed automatically on the server side (Sokolova *et al.*, 2017). Google play store used this technique to scan the Android application before hitting the application market (Penning *et al.*, 2014). Bouncer has potential to take newly uploaded applications to the app market. An instance when the application has the capabilities of sending SMS to the malicious sites or any other criminal activities, such an Android application is classified as malware otherwise benign. However, in this advanced era, it seems that the attackers have found ways to bypass detections. This technique is suitable for Google play store users for the download of applications while the third parties' app store users are not protected with this technique (PlayStore, 2017). Nevertheless, the number of malwares is still growing with the pretty ratio.

The hardest part of the detection of malicious traffic is to differentiate C&C data flow from the normal data flow behaviour (Gu *et al.*, 2009). To overcome this limitation, data mining techniques are very useful to recognize the pattern by extracting the unexpected network patterns (Alparslan *et al.*, 2012). Data mining is the machine learning mostly used to devise methods for classification, prediction, regression, and inference (Eskandari & Hashemi, 2012). These techniques are extensively used in anomaly detection especially in establishing generic and heuristic methods (Schultz *et al.*, 2001). Data mining approaches detect structures in the wide range of data, such as bytecode, and use these structures to detect upcoming malicious occurrences in related data. Gu, Perdisci et al. (2008); Gu et al. (2007); Gu, Zhang et al. (2008); Wang, K. et al. (2011); Yu, X. et al. (2010) proposed BotMiner, BotHunter, BotSniffer, and behaviour-based botnet detection systems respectively based on data mining approach. The techniques are proven to be effective but not without shortcoming (Nagendra Prabhu & Shanthi, 2015). It was experimentally reported that BotMiner, and BotHunter were able to achieve 99% with 1% false alarm, 99.2% with 0.8% false alarm respectively (Zhao, D. et al., 2013).

NEMESYS is a network model-based security solution that combines learning and modelling for detection of anomalies and attacks in the mobile network (Gelenbe et al., 2013). It deals with every mobile connection during communication with each other in a network. The uniqueness of this approach is the difference between the number of mobile users that are monitored and deal in real time are varied (Abdelrahman et al., 2013). Furthermore, a clear and understandable approach was needed to deal with every unique call. Another logic behind constructing this approach was the computational tools that were being developed for anomalies detection were based on mathematical models (Papadopoulos & Tzovaras, 2013). However, NEMESYS is limited to a small number of users and the approach is more complex and memory reserving (Delosières & García, 2013).

2.9 Machine Learning Classifiers

Machine Learning (ML) is extensively used in malware detection, specifically in creating of basic and heuristic methods (Muttik, 2011; Yerima *et al.*, 2013; Yerima *et al.*, 2014b). ML has the capability to generalize the information from huge data sets. In order to apply the generalized information to new actions and solutions it can detects patterns. Supervised and unsupervised are the two main types of machine learning. The class labelled training dataset differentiate the supervised ML from the unsupervised ML (Pedregosa *et al.*, 2011). The algorithm makes decision on the base of this labelled class that is used for training the dataset. Supervised ML has the ability to select the appropriate method depends on the nature of the application. Once the algorithm achieves the acceptable value and level of performance, it stops it's learning. However, the unsupervised machine learning, only demands input data without comparing the yield factors. In this study the different five supervised classifiers are selected on the base of feature length, nature of instances, number of classes, performance and ranking criteria.

2.9.1 Support Vector Machine (SVM)

Support Vector Machine (SVM) is a well-known supervised machine learning technique based on statistical learning theory which is used for nonlinear mapping to convert input data into higher dimensions. Several Machine-Learning (ML) approaches are available for classification of two classes with some of them having exceptional capability for efficient and effective solutions. A linear SVM is adopted for the classification on the basis of used dataset nature, as it is binary (Apvrille, 2012). The main advantage of this technique is in solving the problem of multidimensional and its computational complexity. It can construct a mapping of a non-linear separable data sample by selecting higher dimensional characteristics space. The kernels arrange data instances in a multidimensional space such that they separate two classes of every data instance on a given hyperplane. In our case, giving the weights of the two classes in the training dataset, a hyperplane can separate the classes into malware or botnet with maximum margin. If the accumulated weights are equal to or greater than the prescribed weight of botnet, the class belongs to a botnet, otherwise it will be considered as benign.

The most important idea of using this technique is that every data instance can be classified by a hyperplane if the dataset is transformed into a space with sufficiently high dimensions. In the context of mobile botnet detection, SVM is used to differentiate between benign and botware. Also, it has an accurate detection rate with an acceptable training time. However, most of the existing approaches are independently using this technique for intrusion and malware detection; and either combining or extending them with other ML algorithms. Furthermore, the technique is being used as a feature reduction schemes. Support vector machine is a great approach for intrusion detection systems in artificial neural networks, and SVM generates better results in higher classification accuracy (Mukkamala & Sung, 2002; Sung & Mukkamala, 2003).

2.9.2 J48

The classifier in this study is built as a J48 decision tree algorithm that is based on the C4.5 algorithm. It is designed for the classification of either pruned or unpruned decision tree. The main motive for using J48 is as a result of construction of decision tree from the labelled trained dataset. Once J48 gets the newly arrived dataset by using the dependent and independent variables, it deals with this dataset. In order to make the feature values as a base, it makes decision tree, whereas it found the feature values in the training datasets. Whenever the algorithm encounters a set of items that can clearly be separated from the other class by a specific attribute, it branches out a new leaf according to the value of the attribute. In this process, each time a new decision needs to be taken, the attributes with the highest normalized gain is chosen.

Among all possible values of the attributes, if there are any values for which there is no ambiguity, the branch is terminated, and the appropriate label is assigned to it. The splitting procedure stops when all instances in all subsets belong to the same class. (Shabtai *et al.*, 2014) applied J48 in malware detection and predicted malware class with comparatively high detection rate. A decision tree classifier is used on the basis of these algorithms having an efficient outcome in producing accurate results. Those features which are best in the separating of the botware and benign applications can be clearly seen during the training phase of the decision tree classifier.

2.9.3 Random Forest

Random forest is a combination of bagging method and random subspace proposed by (Breiman, 1996; Ho, 1995). It is a group of classifiers using many decision tree models. In this study, a different subset of training data is selected with a replacement to train each decision tree in this decision tree models. It is very difficult to define the importance of a variable when they interact with other variables, due to its interactions. However, in this case, random forests algorithm estimates its importance by considering the changes which occur in the prediction error of the targeted variable. This calculation is performed by tree according to the random forest construction. The remaining training data serves to estimate the error and variable importance. This classifier is a logic-based algorithm that is proved to produce a high-accuracy result as in (Eskandari & Hashemi, 2012) for malware detection. Random forests have some more benefits over other classifier techniques, such as, it is not over fitting and can run as many trees as it can, 50 times faster than other classification approaches (Ho, 1995). Random forests were chosen on the basis of these upper hands.

2.9.4 Simple Logistic Regression (SLR)

Simple logistic regression is similar to simple linear regression but intended for use with binary outcomes, instead of continuous outcomes. The probability of expected outputs has two values in logistic regression, as aforementioned that it will generate binary output either 0 or 1. It will generate a logistic curve that will remain in the middle of 0 and 1. However, a simple linear regression model is not suitable for predicting the values of two classes of data, as it predicts values from outside of the expected range. (Ng, 2004) proved that L1 and L2 are two types of regularization, with L1 produces a better output when dealing with unrelated and dissimilar vectors of features, while L2 is totally different in this sense. This is the reason for choosing simple logistic regression for this detection technique.

$$\pi = \frac{1}{1+e^{-(\beta_0+\beta_1 X_1+\beta_2 X_2+\cdots+\beta_n X_n)}} \quad 2.1$$

2.9.5 Naïve Bayes

In contrast, this study proposed Naïve Bayes for the prediction of a large number of related applications. In Naïve Bayes, the prediction has an independent assumption which is based on the Bayes Theorem (Pawlak, 2002). The main motive of designing this method was in the use of supervised induction tasks, where the performance was based on the accurate prediction of instances in a test class information (John & Langley, 1995). Classifier could be a straightforward as in applied mathematics formula with a historical record of giving interestingly good result. This is the reason of using it in different malware classification studies (Amos *et al.*, 2013; Khorshed *et al.*, 2012; Yu *et al.*, 2013). It treats numeric and discrete attributes in a different way. Also, Bayes theorem states a way for finding the posterior probability, κ(c|s), from κ(s), κ(c) and κ(s/c). It assumes that there is no relationship among the values of a predictor (s) with a given class (c), with values of other predictors which is called class conditional independence. The probability of a predictor (s) with given class (c) can be depicted from the given Equations 3.3 to 3.5.

$$c \in \{Botware, Benign\} \quad 2.2$$

$$s \in \{Application(s)\ need\ to\ be\ predicted\} \quad 2.3$$

$$\kappa(c|s) = \frac{\kappa(s|c)\kappa(c)}{\kappa(s)} \quad 2.4$$

In the above equations, κ(c|s) is the probability of class c, with the predictor x, κ(c) is the next probability of class c, κ(s|c) is the probability of the predictor (s) with class (c), moreover, κ(s) is the last probability of the predictor (s).

2.10 Discussion

In the related works mentioned above, open issues that are concerned with the progressive security of smartphones against botnet attacks were identified. The various existing challenges are highlighted with all due respect to Android botnet that need to be addressed by the researchers alike:

The existing malware detection systems target on mobile malware analysis and detection in general (Peiravian & Zhu, 2013; Rashidi & Fung, 2016; Sanz *et al.*, 2013; Yerima *et al.*, 2014a). Therefore, no such detection technique focusses on Android third party's applications that are involved in botnet activities. These activities can be differentiated from the malware activities by the action performed on the basis of instructions received from the botmaster. The detail about these activities is available in above sections. Thus, Android botnet detection is a novelty research area that needs to be addressed.

Android Features: Android features are grouped into static and dynamic. Permissions are the most used static features. Permissions are further grouped in normal, dangerous, signatures and signaturesOrsystem. Few of the work used API Calls for botnet detection. However, the Activities, broadcast receivers and services are also used by each Android application. These features are declared in the AndroidManifest.XML file. In this work all, these static features are chosen due to its potential and richness in detection of Android botnet.

Currently, there are millions of third party's Android applications available in online market. Thus, it is a challenging task to perform botnet analysis on a huge number of Android applications with varied features space. Therefore, selecting the most related static features of Android applications with botnet facilities is a pivotal task.

Cross-functional activities: Initially, it is mandatory to activate the cross-functional group. It should involve researchers either from industries or institutional and stakeholders namely, enterprises, networks, Internet service providers and governments for the recognition of Android botnets and potential distraints of botnet tools. A clear policy on smartphones usage must be mentioned and standardized across the enterprise. Moreover, the smartphones users should give awareness about ways by which an Android botnet attack can be solved.

Some of the existing solutions that are based on dynamic analysis require computational-intensive resources, code coverage, and processing time. Consequently, it is infeasible to apply dynamic analysis to huge datasets. Instead, static analysis results provide more insights into the coding patterns of an Android application. This approach is considered the lightweight detection and analysis option.

Considering the enumerated open research challenges for botnet detection and analysis in mobile devices, static analysis and detection approach is extremely important. Such an approach should deal with botnet susceptible features of Android applications during installation time.

2.11 Summary

This chapter summarized the importance of mobile devices that have become similar to the personal computer in terms of processing, communication, storage and other functionalities. Moreover, the security of mobile devices is also summarized which is the most challenging issue facing by researchers. Specifically, Android botnet is the current existing dangerous threat facing by mobile devices and can have many consequences if ignored. Their capabilities were confirmed by exploring their definition and history. The potential threats of Android botnets were briefly discussed. Furthermore, its components, architecture, and design are explored, while from this literature it was found that the main target of Android botnets is C&C channels. In this chapter, many of the Android botnet detection techniques are investigated and were found to follow any method among dynamic, hybrid or static approaches. From the review of literature, it was discovered that the existing techniques have limitations with regard to the progressive security of Android devices against the botnets. Furthermore, state-of-the-art from the literature review is presented as they will serve as a roadmap for researchers. Moreover, different open challenges are highlighted with respect to Android botnets which need to be addressed by the researchers either from institutions or industries.

CHAPTER 3

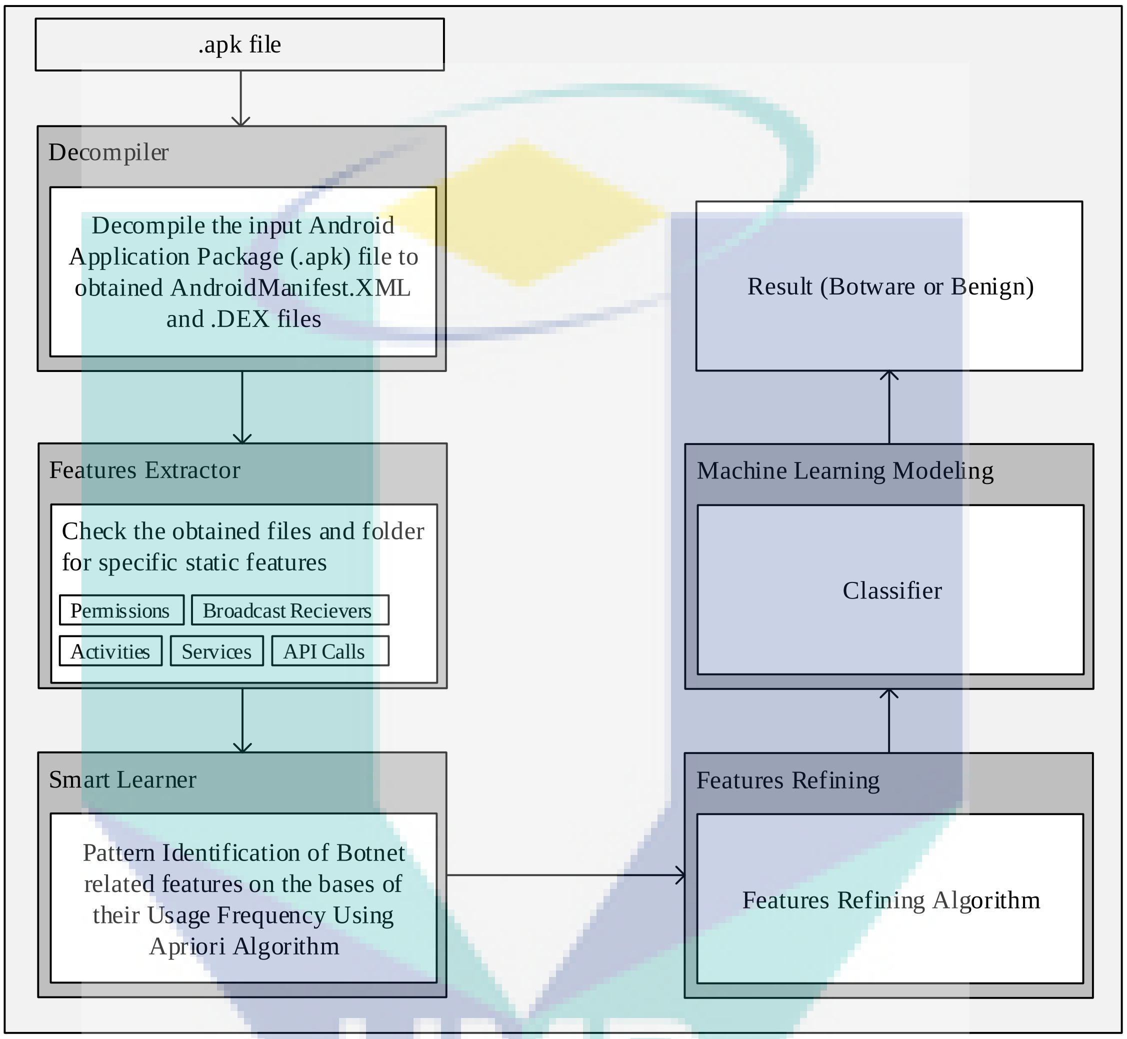
AN ENHANCED ANDROID BOTNET DETECTION APPROACH USING FEATURE REFINEMENT

3.1 Overview

The previous chapter discussed research work related to this study. These studies used different tools to analyse and conduct their experiments. The familiarity with the existing tools intensively increases the knowledge of malware analysis techniques. This chapter aims to present the details of the proposed enhanced Android botnet detection approach using feature refinement. The building blocks and components of the proposed detection approach with their functionality are described. This approach is comprised of five main components namely: decompiler, features extractor, smart learner, features refining, and machine learning modelling. The remaining chapter is divided into three sub-sections. Section 3.2 illustrates the proposed approach while the main components of the proposed approach are described in Section 3.3. This chapter is concluded with Section 3.4 to summarize the whole approach.

3.2 The Proposed Approach

In this sub-section, overview of the Android detection approach was presented. The proposed approach has five main components as shown in Figure 3.1. The first component is the decompiler which is responsible for dissecting the APK file and decoding its components. Every Android application has different directories such as AndroidManiFest.XML, asset, DEX, and resources. These are mentioned with details in Chapter 2. The second component is the extractor which, take AndroidManiFest.XML, and DEX classes as an input from the decompiler component. In this component, the static features namely: permissions, activities, broadcast receivers, and services are extracted from AndroidManiFest.XML. However, the API calls are extracted from DEX classes. All the extracted features are represented with binary numbers. The existing feature is represented with “1” while the absent one is represented with “0” for further processing.



.apk file
Decompiler
Decompile the input Android Application Package (.apk) file to obtained AndroidManifest.XML and .DEX files
Features Extractor
Check the obtained files and folder for specific static features
Permissions
Broadcast Recievers
Activities
Services
API Calls
Smart Learner
Pattern Identification of Botnet related features on the bases of their Usage Frequency Using Apriori Algorithm
Features Refining
Features Refining Algorithm
Machine Learning Modeling
Classifier
Result (Botware or Benign)

Figure 3.1 Android Botnet Detection Proposed Approach

In the third component, all the available features are indexed and the Apriori algorithm was applied on the extracted indexed features in order to identify the frequent used features. The fourth component is feature refining; in this component, the identified features are then refined with the help of Information Gain (IG) algorithm to select the most used features on the base of their frequencies (Uğuz, 2011). Also, in this component, the features which are botnet susceptible are refined. The fifth and final component is the machine learning modelling which classified the input android applications on the bases of their used features.

3.2.1 Characteristics of proposed approach

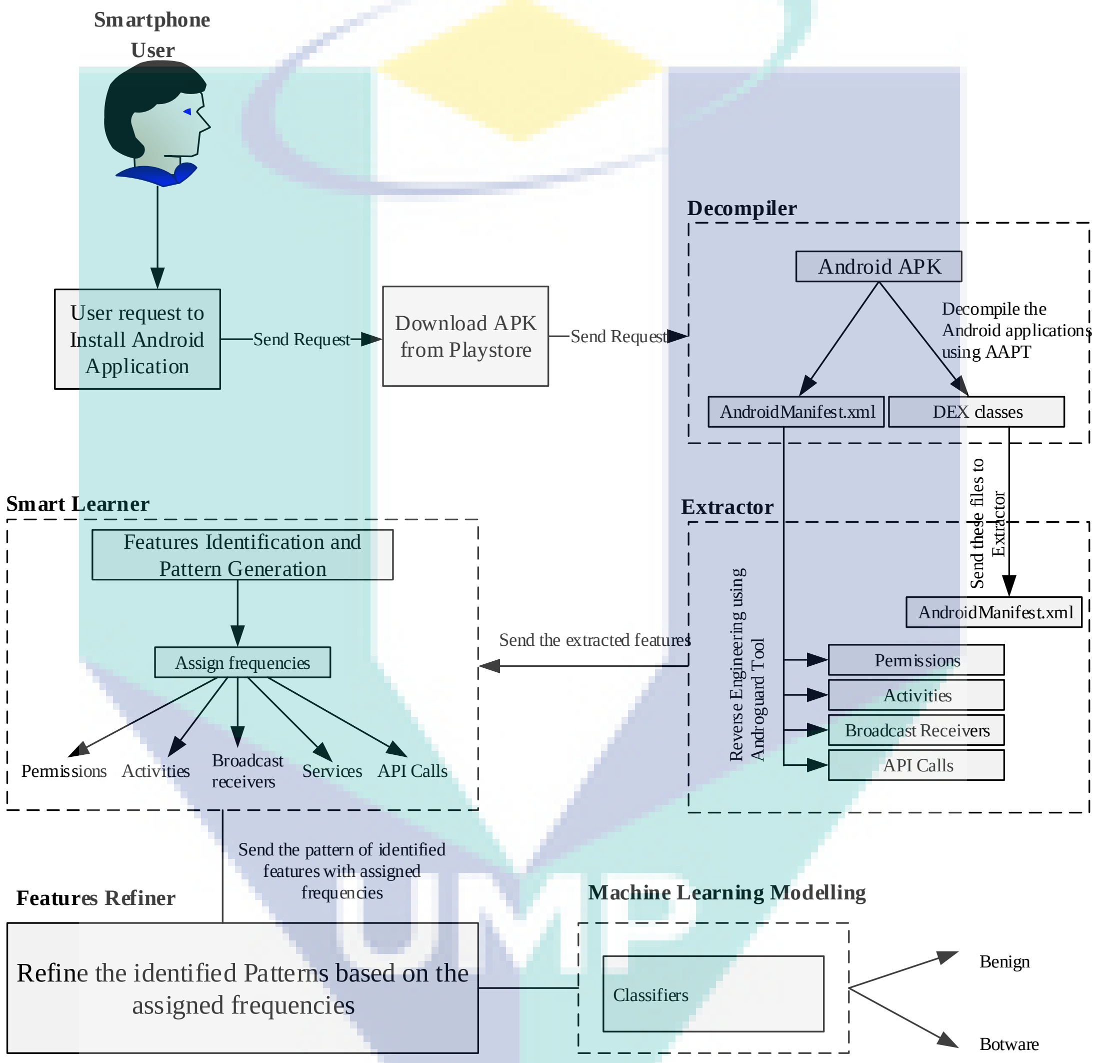
As stated before, the aim of the proposed approach is to detect botnet attacks in Android devices. Thus, the proposed detection approach has the following characteristics that distinguish it from the existing botnet detection techniques:

- **Generalizable**: The fundamental cause of any type of botnet attacks in the Android devices is the openness nature of Android OS. However, this approach extracts all types of features of Android applications by using additional refining component during the installation on smartphones. Therefore, the proposed approach has the ability to detect any type of botnet attacks in Android smartphones and is generalizable.
- **Portability**: Android OS can be installed almost on every smartphone; hence, this detection approach is based on Android OS. Therefore, the proposed approach can be implemented on every Android smartphone.
- **Saving Power and Computation**: The proposed approach is based on the static features; hence, it requires less amount of battery and computation power. However, the dynamic features-based detection approach requires an isolated environment to run the third-party Android applications. So therefore, it requires more amount of computation and battery power. This leads to the improvement in the overall performance.
- **Scalable**: This approach uses the machine learning model to classify the applications as botware or benign. The user may increase the number of applications in the dataset and can be able to re-train the approach by updating the machine learning algorithm. This led the proposed approach to more powerful for the botnet detection in Android OS.
- **Improved Accuracy**: With the addition of new component namely features refining component and new additional static features which include activities, broadcast receivers, and services to the detection approach, the accuracy of the proposed approach is greatly improved.

3.3 Components of proposed approach

This section describes the key components of the proposed approach namely: decompiler, features extractor, smart learner, features refining, and machine learning modelling. The first component is responsible for analyzing the Android applications and decoding its main files. The Android applications contain different major components.

However, AndroidManifest.xml is the most important file that must be included in each Android application to be decompiled. Similarly, the DEX file is the next important file that needs to be decompiled in this step. The second component in this proposed approach is responsible for features extraction namely: permissions, activities, broadcast receivers, and services extraction from AndroidManifest.xml while the API calls are extracted from the DEX classes. Figure 3.2 shows the execution of the proposed approach.



Smartphone User
User request to Install Android Application
Send Request
Download APK from Playstore
Send Request
Decompiler
Android APK
Decompile the Android applications using AAPT
AndroidManifest.xml
DEX classes
Send these files to Extractor
Extractor
Reverse Engineering using Androguard Tool
AndroidManifest.xml
Permissions
Activities
Broadcast Receivers
API Calls
Send the extracted features
Smart Learner
Features Identification and Pattern Generation
Assign frequencies
Permissions
Activities
Broadcast receivers
Services
API Calls
Send the pattern of identified features with assigned frequencies
Features Refiner
Refine the identified Patterns based on the assigned frequencies
Machine Learning Modelling
Classifiers
Benign
Botware

Figure 3.2 Flow of the Proposed Botnet Detection Approach Using Static Features Analysis

In the third component, all the used features are identified and indexed for easy understanding. As a result of this process, all the identified and indexed features are then represented as a single instance with binary representation and a class label. If an examined application has the feature called INTERNET, this should be represented with 1 and if absent then 0. Then the features are grouped in the CSV file for further analysis process. The fourth component is known as features refining, whereby all the features that has C&C features associated are examined. The machine learning modelling is the fifth component. It will examine the application for botware or benign on the base of used features. The details of each component of the proposed approach are given in the following sub-sections.

3.3.1 Decompiler

In this sub-section, decompiler component of proposed Android botnet detection approach is described. Decompiler is responsible for dissecting the Android application to decode its main components. For this purpose, botware and benign applications are collected from the online repositories, including Google Play Store, Contagio malware repository and the well-known Drebin dataset (Arp *et al.*, 2014; Parkour, 2011; PlayStore, 2017). In total, 3535 botware and 3500 benign samples are selected for initial analysis. In the first phase, APKtool is used to decompile the selected applications to obtain AndroidManifest.xml and DEX classes. APKtool was preferred for since it utilizes the recent Android SDK, which is a better approach for files optimization (Kang *et al.*, 2014; Winsniewski, 2012). Besides the open accessibility, it has the ability to decode resources almost to its original shape (Faris, 2017). Additionally, it correctly decodes the AndroidManifest.xml and disassembles the DEX files. AndroidManifest file should be in the root director of each application. It contains the essential information and components that represent an app to the Android system. This includes name, version, and components, such as activities, services, broadcast receivers, contents providers, and access rights of the APK. Furthermore, it describes the messages intents that can be handled, and also determine process that will host components. Permissions which declare the security of an Android application are also declared in this file.

The basic structure of AndroidManifest.xml and Dex files are given in Figure 3.3 and Figure 3.4 respectively. Although, this file may not be completely java code, but still, it is readable. The output of this component is the input to the feature's extractor.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="https://www.odarzi.com/apk/res/android"
Package="com.androidapp.odarzibasicElements"
Android:versionCode="1"
Android:versionName="1.0">
<application>
<activity android:name="odarzibasicElements">
  <intent-filter>
     <action android:name="android.intent.action.MAIN"/>
     <category android:name="android.intent.category.LAUNCHER"/>
  </intent-filter>
</activity>
<activity-alias>
     <intent-filter> . . . </intent-filter>
     <meta-data />
</activity-alias>
</application>
<uses-sdk android:minSdkVersion="2" />
</manifest>
```

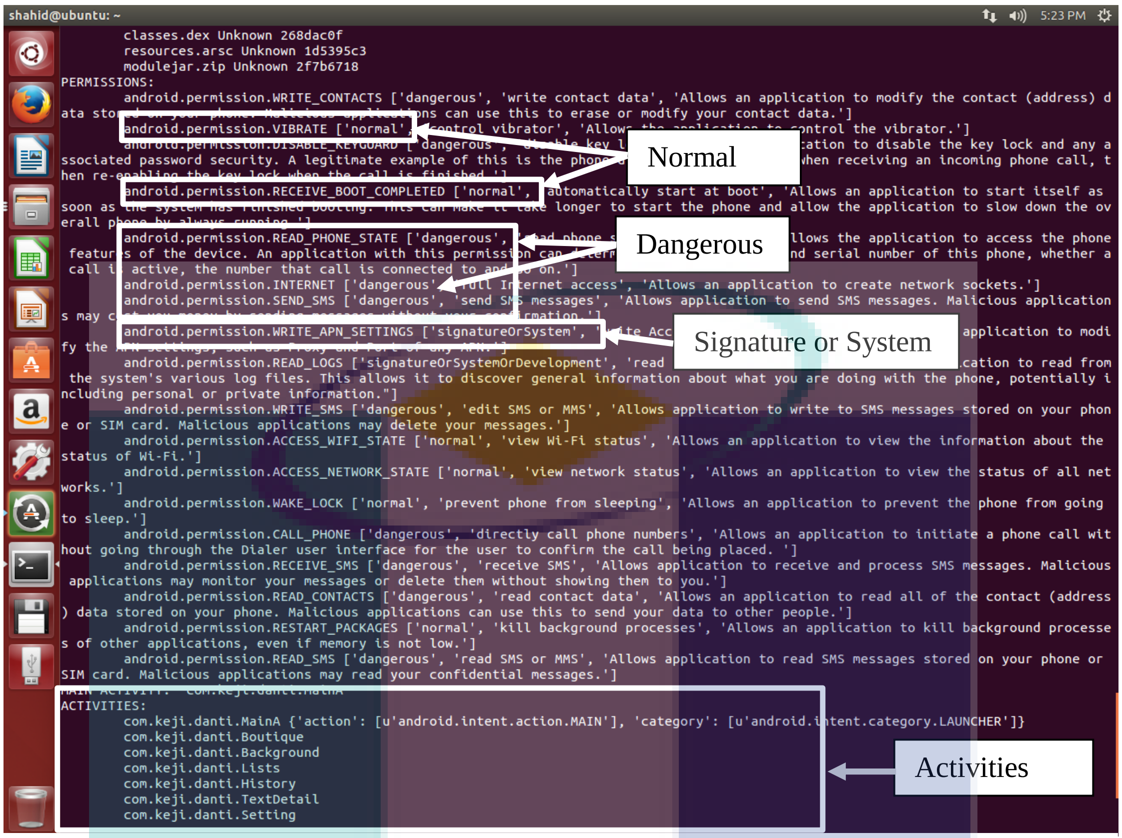
Figure 3.3 AndroidManifest.XML file structure

The DEX file is made up of several sections where Figure 3.4 outlines the most important ones with respect to application analysis. This file holds the APK resources, however, result of any modification in these files will directly affect the APK.

```
Dexfile {
      header       header_item,
      string_ids   string_id_item[],
      type_ids     type_id_item[],
      proto_ids    proto_id_item[],
      field_ids    field_id_item[],
      method_ids   method_id_item[],
      class_defs   class_def_item[],
      data         ubyte[],
      link_data    ubyte[]
}
```

Figure 3.4 DEX File Structure

Figure 3.5 shows the example of decompiled features from an Android application using Androguard tool. This example shows that these features exist in the input application. Permissions feature is highlighted which is further categorized into four sub-categories according to their nature as given in Section 2.5.



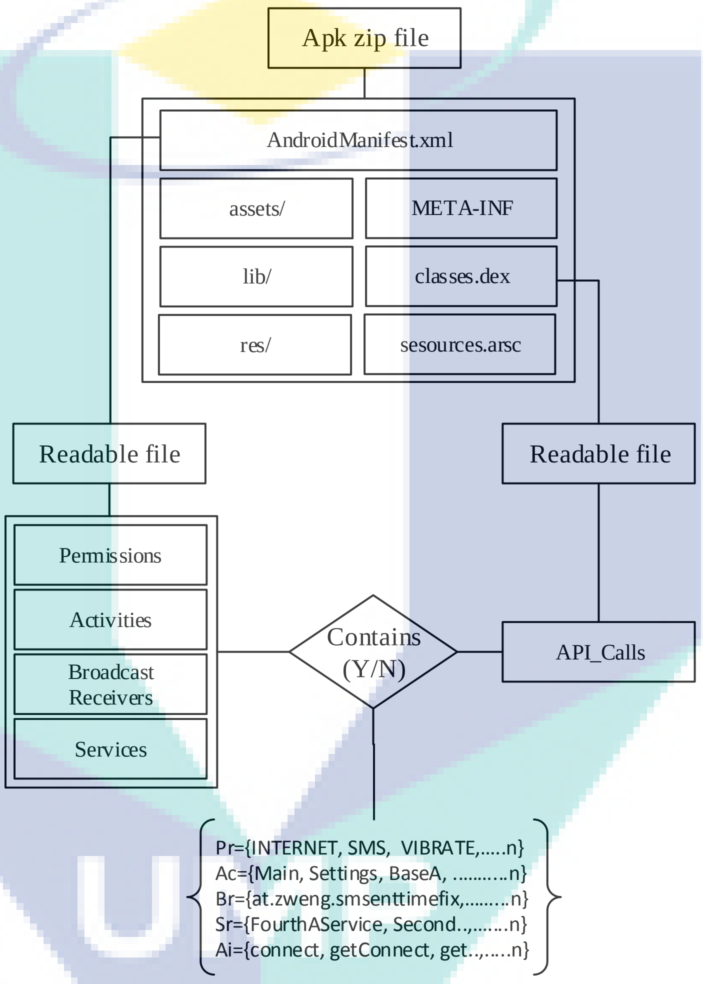
```
        classes.dex Unknown 268dac0f
        resources.arsc Unknown 1d5395c3
        modulejar.zip Unknown 2f7b6718
PERMISSIONS:
        android.permission.WRITE_CONTACTS ['dangerous', 'write contact data', 'Allows an application to modify the contact (address) d
ata sto[illegible]ns can use this to erase or modify your contact data.']
        android.permission.VIBRATE ['normal', [illegible]ontrol vibrator', 'Allow[illegible]ontrol the vibrator.']
        [illegible] ['dangerous', [illegible]ation to disable the key lock and any a
ssociated password security. A legitimate example of this is the phone [illegible]when receiving an incoming phone call, t
hen re-e[illegible]
        android.permission.RECEIVE_BOOT_COMPLETED ['normal', 'automatically start at boot', 'Allows an application to start itself as
soon as [illegible] longer to start the phone and allow the application to slow down the ov
erall p[illegible]
        android.permission.READ_PHONE_STATE ['dangerous', [illegible]llows the application to access the phone
 featur[illegible]s of the device. An application with this permissi[illegible]nd serial number of this phone, whether a
 call i[illegible] active, the number that call is connected to and[illegible]o on.']
        android.permission.INTERNET ['dangerous', [illegible]ernet access', 'Allows an application to create network sockets.']
        android.permission.SEND_SMS ['dangerous', 'send SM[illegible] messages', 'Allows application to send SMS messages. Malicious application
s may c[illegible]rmation.']
        android.permission.WRITE_APN_SETTINGS ['signatureOrSystem', '[illegible]ite Acc[illegible] application to modi
fy the [illegible]
        android.permission.READ_LOGS ['signatureOrSystemOrDevelopment', 'read [illegible]cation to read from
 the system's various log files. This allows it to discover general information about what you are doing with the phone, potentially i
ncluding personal or private information."]
        android.permission.WRITE_SMS ['dangerous', 'edit SMS or MMS', 'Allows application to write to SMS messages stored on your phon
e or SIM card. Malicious applications may delete your messages.']
        android.permission.ACCESS_WIFI_STATE ['normal', 'view Wi-Fi status', 'Allows an application to view the information about the
status of Wi-Fi.']
        android.permission.ACCESS_NETWORK_STATE ['normal', 'view network status', 'Allows an application to view the status of all net
works.']
        android.permission.WAKE_LOCK ['normal', 'prevent phone from sleeping', 'Allows an application to prevent the phone from going
to sleep.']
        android.permission.CALL_PHONE ['dangerous', 'directly call phone numbers', 'Allows an application to initiate a phone call wit
hout going through the Dialer user interface for the user to confirm the call being placed. ']
        android.permission.RECEIVE_SMS ['dangerous', 'receive SMS', 'Allows application to receive and process SMS messages. Malicious
 applications may monitor your messages or delete them without showing them to you.']
        android.permission.READ_CONTACTS ['dangerous', 'read contact data', 'Allows an application to read all of the contact (address
) data stored on your phone. Malicious applications can use this to send your data to other people.']
        android.permission.RESTART_PACKAGES ['normal', 'kill background processes', 'Allows an application to kill background processe
s of other applications, even if memory is not low.']
        android.permission.READ_SMS ['dangerous', 'read SMS or MMS', 'Allows application to read SMS messages stored on your phone or
SIM card. Malicious applications may read your confidential messages.']
MAIN ACTIVITY:  com.keji.danti.MainA
ACTIVITIES:
        com.keji.danti.MainA {'action': [u'android.intent.action.MAIN'], 'category': [u'android.intent.category.LAUNCHER']}
        com.keji.danti.Boutique
        com.keji.danti.Background
        com.keji.danti.Lists
        com.keji.danti.History
        com.keji.danti.TextDetail
        com.keji.danti.Setting
```

Figure 3.5 Android Application features extraction Using Android SDK Tool

3.3.2 Features Extractor

This is the second component of the proposed approach which takes input from the decompiler. In this component, reverse engineering is performed on each input application so as to extract static features from the AndroidManifest.xml and DEX file. Aforementioned was that AndroidManifest file contains the important features of an application. Besides, this file contains tags to interact with these features inside and outside of the Android applications. For example, the tag *<services…>, <receiver …> and <uses-permission>* specified software and hardware requirements. The features tag provides some extra mandatory information to the permissions that help with a behavioral interpretation of the analyzed applications. For instance, the permission feature, namely RECEIVE_BOOT_COMPLETED, is undeclared added to the AndroidManifest file once an application requests permission to use UPDATE functionality. The tag *<uses-permissions>* is used to request a set of permissions that the application requires in order to install correctly and access the private parts of mobile devices and application. This study revealed that there are certain combinations of requested permissions and used features while performing malicious activities. For instance, a successfully installed application that requests the INTERNET, READ_PHONE_STATE, and

READ_EXTERNAL_MEMORY permissions may collect smartphone-related information and send this information to a botmaster. It can only interact with the smartphones when the user grants permissions to the applications during installation or later on. However, all the requested permissions may or may not be used by the applications. Similarly, other mentioned features may or may not be used by the applications.



Apk zip file
AndroidManifest.xml
assets/
META-INF
lib/
classes.dex
res/
sesources.arsc
Readable file
Readable file
Permissions
Activities
Broadcast Receivers
Services
Contains (Y/N)
API_Calls
Pr={INTERNET, SMS, VIBRATE,.....n}
Ac={Main, Settings, BaseA, ...n}
Br={at.zweng.smsenttimefix,........n}
Sr={FourthAService, Second..,.......n}
Ai={connect, getConnect, get..,.....n}

Figure 3.6 Feature Extraction from AndroidManifest and DEX file

In order to extract these features from each Android application, Androguard tool is used (Desnos, 2011b), which is an open source project and made available to the public. Figure 3.6 shows the feature extraction process from the AndroidManifest and DEX file. The static features namely: permissions, activities, broadcast receivers, services and API calls are extracted from the AndroidManifest.XML and DEX classes which are given in Sections 2.4. In order to perform the extraction of features automatically a Python code is applied to all Android applications. Once the extraction of all features is completed, all the extracted features are stored in the CSV files for further analysis.

Let *l* and *m* be the number of Android applications and the set of features including permissions, activities, broadcast receivers, services and API calls. The features vector for application *i* is ($x_{i,1}$, $x_{i,2}$, $x_{1,3}$,…, $x_{i,j}$) where:

$$X(l,k) = \begin{cases} 1 & if\ application\ l\ uses\ feature\ k \\ 0 & otherwise \end{cases} \qquad 3.1$$

Similarly, suppose a class of an instance in the generated dataset is ci ∈ (Botnet & Benign) which shows the class of an application i (Kheir et al., 2014). Formally, each extracted application is saved in the CSV formatted file for further analysis. The static features namely: permissions, activities, broadcast receivers, services and API calls are explained in the following sub-sections.

$$P(x_i|y_i) = c_i|x_{i,1}, x_{i,2}, x_{i,3}, \dots \dots \dots x_{i,j}, = \prod_{k=1}^{j} P(x_i|y_{i,k}) \qquad 3.2$$

The example of CSV file(s) is shown in Figure 3.7. The file begins with the hash function of the application and ends with the sum of all the enabled features. The hash value represents the MD5 values of the Android applications. However, the values “1” and “0” correspond to enabled and disabled features, respectively. The sum of these enabled values is utilized for the further analysis. The phenomenon shows that applications that are using more features pretend to have botware intention.

```
(00DA00BA346A4B1AB452651A003A0BA37A463E4A4BAB452651A),<1,1,
1,0,1,1,0,0,1,0,1,0,1,1,1,1,1,1,1,0,0,1,0,1,0,1,0,0,0,1,1,1,1,1,0,0,0,1,0,1,0,1,0,1,0,
1,1,1,1,0,1,1,0,0,1,0,1,0,1,1,1,1,1,1,1,0,0,1,0,1,0,1,0,0,0,1,1,1,1,1,0,0,0,1,0,1,0,1,
0,1,0,11,1,1,0,1,1,0,0,1,0,1,0,1,1,1,1,1,1,1,0,0,1,0,1,0,1,0,0,0,1,1,1,1,1,0,0,0,1,0,
0,1,1,1,1,1,1,0,0,1,0,1,1,1,>
```

Figure 3.7 Structure of CSV file

Table 3.1 tabulates the top 20 permissions used by botware and benign applications. These permissions features are listed in descending order according to their usage. The table depicts that the INTERNET permission is used by 84.29 % of the botware application while 55.71% by the benign applications. Similarly READ_PHONE_STATE is used by 82.14% of the botware applications, RECEIVE_BOOT_COMPLETED is used by 80 % while the39.29% and 37.86% respectively used by benign applications. Furthermore, the enormous difference is seen in the SEND_SMS, RECEIVE_SMS, and READ_SMS that is 79.29%, 78.57% and

74.29% used by botware applications while the 4.29%, 11.43% and 6.43% used by benign applications respectively.

Table 3.1 Top 20 Used Permission Features with their Percentage (%)

S. No	Permissions	Botware	Benign
1	INTERNET	98.00	78.10
2	READ_PHONE_STATE	94.40	77.20
3	RECEIVE_BOOT_COMPLETED	89.70	61.10
4	SEND_SMS	86.80	60.50
5	READ_SMS	85.10	60.10
6	WAKE_LOCK	79.70	55.30
7	RECEIVE_SMS	74.30	41.20
8	READ_CONTACTS	73.90	41.10
9	ACCESS_WIFI_STATE	65.80	37.60
10	VIBRATE	65.40	34.30
11	CALL_PHONE	64.30	10.00
12	WRITE_SETTINGS	63.60	30.70
13	WRITE_SMS	63.60	10.00
14	WRITE_CONTACTS	61.40	10.70
15	WRITE_EXTERNAL_STORAGE	60.00	08.60
16	ACCESS_FINE_LOCATION	59.30	12.90
17	CHANGE_WIFI_STATE	57.90	30.00
18	GET_TASKS	55.00	12.10
19	SYSTEM_ALERT_WINDOW	41.40	08.60
20	ACCESS_NETWORK_STATE	41.20	08.60

For easy understanding, all of the extracted features are indexed such as INTERNET is indexed with P1, READ_PHONE_STATE with P2, RECEIVE_BOOT_COMPLETED with P3 and much more. Table 3.2 shows the indexed permissions features. However, the activities, broadcast receivers, services, and API call features indices are given in Appendix B.

Table 3.2 Top 20 Used Permission Features Index

S. No	Permissions	PID
1	INTERNET	P1
2	READ_PHONE_STATE	P2
3	RECEIVE_BOOT_COMPLETED	P3
4	SEND_SMS	P4
5	READ_SMS	P5
6	WAKE_LOCK	P6
7	RECEIVE_SMS	P7
8	READ_CONTACTS	P8
9	ACCESS_WIFI_STATE	P9
10	VIBRATE	P10
11	CALL_PHONE	P11
12	WRITE_SETTINGS	P12

Table 3.2 Continued

13	**WRITE_SMS**	**P13**
14	WRITE_CONTACTS	P14
15	WRITE_EXTERNAL_STORAGE	P15
16	ACCESS_FINE_LOCATION	P16
17	CHANGE_WIFI_STATE	P17
18	GET_TASKS	P18
19	SYSTEM_ALERT_WINDOW	P19
20	ACCESS_NETWORK_STATE	P20

3.3.3 Smart Learner

This component takes the input from the feature's extractor. The main function of the smart learner is to analyze different group of benign and botware applications in order to identify the unique pattern of features which are susceptible to botnet attacks. For pattern identification the value of each feature must be known in advance. In order to calculate the value of each feature the smart learner count the number of each feature occurrence in the benign and botware applications. The feature occurrence is calculated by using the Equation 3.3. Once the used features occurrence is calculated then these values are assigned to all the extracted features. These values are used to calculate the percentage of each feature in both categories. All the applications in both categories are analyzed in terms of used and requested features. After the inspection of selected samples, most prominent features are counted.

$$Fn = \frac{\sum_{i=1}^{n} Pi}{NBt} \tag{3.3}$$

$$Fn = \frac{\sum_{i=1}^{n} Pi}{NBn} \tag{3.4}$$

In the Equation 3.3, Fn represent the specific permission, while the Pi represent the total number of n^{th} permission occurrence in the NB_t, where NB_t represent the total number of selected botware applications. The same formula is applied for activities, broadcast receivers, services and API calls features. Similarly, the occurrences of aforementioned features are calculated using Equation 3.4. Where NB_n represent the total number of benign applications in the dataset. Table 3.3 shows the calculated values of top 20 requested permissions by botware and benign applications. From the table, it seems that botware applications requested more permissions as compared to benign ones. The same process is performed for other mentioned features and the calculated values of these features are given in Section 4.3.

Table 3.3 Frequency of Top 20 requested permissions by botware and benign applications

S. No	Permissions	Botware	Benign
1	P1	0.98	0.78
2	P2	0.94	0.77
3	P3	0.90	0.61
4	P4	0.87	0.61
5	P5	0.85	0.60
6	P6	0.80	0.55
7	P7	0.74	0.41
8	P8	0.74	0.41
9	P9	0.66	0.38
10	P10	0.65	0.34
11	P11	0.64	0.10
12	P12	0.64	0.31
13	P13	0.64	0.10
14	P14	0.61	0.11
15	P15	0.60	0.09
16	P16	0.59	0.13
17	P17	0.58	0.30
18	P18	0.55	0.12
19	P19	0.41	0.09
20	P20	0.41	0.09

Once all the values for selected features are calculated. Based on these values smart learner generate pattern by using the Apriori algorithm. WEKA tool is used for this process (Agrawal, Imielinski, *et al.*, 1993; Agrawal, Imieliński, *et al.*, 1993; Agrawal & Srikant, 1994; Hall *et al.*, 2009). The Apriori algorithm was chosen to identify the pattern of significant features combination because it has been regularly and successfully used for existing problems (Smith & Frank, 2016). This algorithm deals with the subset of events beyond examining the specific order of events. The Apriori algorithm takes dataset DB_t as an input that contains full set of used features of *n* botnet applications. Let $I = \{P1, P2, \ldots Pn, A1, A2, \ldots. An, B1, B2, \ldots Bn, S1, S2, \ldots. Sn, AP1, AP2, \ldots APn\}$ be an instance of DB_t. The Apriori algorithm begins by pinpointing the individual repeated items in the DB_t dataset and extending them to substantial sets of items as much those item sets sufficiently appear often in the aforementioned dataset. For example, $A= \{P_1, A_1, B_1, S_1, AP_1\}$ be a candidate item set. There are two values need to be known in advance for the Apriori algorithm which are support and confidence for the calculation of the frequency of features used in the DB_t dataset. In this case, the support value of the candidate item set $\{P_1, A_1, B_1, S_1, AP_1\}$ is computed as given below.

$$Support\ (P1, A1, B1, S1, AP1) = \frac{\text{Number of applications that contains } P1{,}A1{,}B1{,}S1{,}AP1 \text{ in } DB_t}{Complete\ set\ of\ Applicaitons\ in\ DB_t} \quad 3.5$$

The candidate item set is considered as a frequent item set or a relevant pattern, only if $support\ (P1, A1, B1, S1, AP1) \geq \text{threshold (t)}$, where t is a user-defined minimum threshold. In this study, we set 0.5 as a minimum support threshold. However, the same process is applied for frequent item set identification for benign applications.

Figure 3.8 describes the smart learner algorithm. This takes extracted features, DB_t and DB_n as input from the feature extractor component and set a threshold value. DB_t is the number of total applications in Botware dataset whereas DB_n is the total number of applications in the benign dataset. As explained earlier, it calculates the support value for each pattern based on the assigned value by using the Apriori algorithm. The given algorithm generates botware and benign based on the generated pattern. The generated pattern will be botware if the features usage frequency in that unique pattern is greater than or equal to the threshold value, otherwise it will be benign.

Figure 3.8 Smart Learner Algorithm

```
Input: Extracted Features, $\boldsymbol{DB_t}$, $\boldsymbol{DB_n}$
1:  Calculate the Occurrence frequency of each feature in $\boldsymbol{DB_t}$ and $\boldsymbol{DB_n}$
2:  $\boldsymbol{Fn} = \frac{\sum_{i=1}^{n} Fi}{NBt}$        where n the nth feature in $\boldsymbol{DB_t}$
3:  $\boldsymbol{Fn} = \frac{\sum_{i=1}^{n} Fi}{NBn}$        where n the nth feature in $\boldsymbol{DB_n}$
4:  Calculate support value
    Support (Pi,Ai,Bi,Si,APi)= (Number of Applications that contains Pi, Ai, Bi, Si,
                               APi in a $\boldsymbol{DB_t}$) / (Total number of Applications in
                               $\boldsymbol{DB_t}$)
5:          Pattern ← Support
6:          Set a Threshold value equal to 0.5
7:          If Pattern > = Threshold value
8:                  Pattern ← Botware
9             Else
10:                 Pattern ← Benign
11:           End if
```

The identified unique patterns for botware and benign applications are shown in Table 3.4. The pattern ID represents the indexed ID of unique features pattern, while the support values are calculated for each unique feature patterns. The complete list of unique patterns is given in Appendix C. However, Table 3.4 listed the top 40 unique patterns with their support values. Table 3.4 depicts that the botware applications utilize the combination of features for malicious activities, such as, INTERNET, RECEIVE_SMS, WRITE_EXTERNAL_STORAGE, com.clientsoftware.ServiceStartr, com.phone.call-corexy.xy.SReceiver, and SYSTEM_ALERT_WINDOW perform the malicious activities on the smartphone and steal sensitive information from it. Once this mentioned malicious activity is performed, it sends the stealth information to the C&C server through the communication channel. In this malicious activity INTERNET permission provide the connection between smartphone and C&C server, while the RECEIVE_SMS permission received the updates and commands about the activity. It is reported that sending of SMS and MMS to the premium numbers can cause financial losses (Johnson & Traore, 2015). Botware applications having these INTERNET, WRITE_SMS and SEND_SMS permissions enable, can send SMS and MMS to premium numbers with the combination of MAIN ACTIVITY and TOUCHSCREEN.

Furthermore, the location related to permissions such as ACCESS_COARS-_LOCATION and ACCESS_FINE_LOCATION is routinely used for the smartphone information collection and network location data gathering. The pattern of UP29 is {P_1, P_{20}, B_{12}, AP_3} which is the combination of INTERNET, ACCESS_NETWORK_STA-TE, com.google.android.mms.LiveReceiver and com.clie-ntsoftware.SDCardServiceSt-arter are used to handle the connection between bots and botnets.

Table 3.4 Top 40 Unique Pattern for Botware and Benign Used Features

Unique Pattern ID	**Used features Pattern**	**Support Values**	
		Botware	**Benign**
UP1	{P1,P7,P17,B4,AP6,P19}	0.9731	0.0269
UP2	{P1,P5,P19,B4,AP19,S15,AP6}	0.9374	0.1059
UP3	{P1,P15,B11,AP17,P18,S10}	0.9151	0.0773
UP4	{P1,P5,P10,B6,AP6,S15,AP6,P5,A18}	0.9045	0.1370
UP5	{P1,P14,B18,AP16,S7}	0.9009	0.1831
UP6	{P1,P12,B20,AP9,S5,AP17}	0.8856	0.0000
UP7	{P1,P16,B17,AP7,S11,AP1,P2,S11}	0.8806	0.1059
UP8	{P1,P2,B19,AP15,S12,AP5,P16,S19,S7}	0.7949	0.0731
UP9	{P1,P20,B2,AP18,S8,AP8,P18}	0.7867	0.1363
UP10	{P1,P14,B10,AP12,S14,AP2,P7,S5}	0.7796	0.0235
UP11	{P1,P20,B16,AP14,S4,AP13,P14,A15}	0.7774	0.1831
UP12	{P1,P6,B16,AP9,S4,AP13,P6,S9,S2,A15	0.7758	0.2055
UP13	{P1,P10,B3,AP5,S20,AP19,P3}	0.7712	0.1831
UP14	{P1,P8,S1,AP10,S12,AP5,P12}	0.7705	0.0216
UP15	{P1,P5,B5,AP15,S19,AP14,P16,S16}	0.7349	0.0000
UP16	{P1,P7,S1,AP14,S5,AP17}	0.722	0.0000
UP17	{P1,P4,B8,AP1,S6,AP18,P10,S17,S11}	0.7214	0.0000

Table 3.4 Continued

Unique Pattern ID	Used features Pattern	Support Values	
		Botware	Botware
UP18	{P1,P13,B9,AP11,S11,AP1}	0.7202	0.0216
UP19	{P1,P6,B7,AP20}	0.7178	0.1945
UP20	{P1,P9,B13,AP4,S20,AP19,P11,S13}	0.7173	0.0731
UP21	{P1,P5,B17,AP20,S7,AP15,P15,S3}	0.7089	0.0000
UP22	{P1,P18,B14,AP1,S10,AP3,P10}	0.7015	0.0556
UP23	{P1,P13,B3,AP16,S18,AP12,P13,S9,S13}	0.6974	0.0000
UP24	{P1,P17,B13,AP17,S10,AP3,P1,S18}	0.6828	0.0773
UP25	{P1,P20,B20,AP11,B1,AP4,P9,S12,S17}	0.6773	0.0000
UP26	{P1,P4,B12,AP6,S3}	0.6731	0.1945
UP27	{P1,P11,B5,AP2,S8}	0.6544	0.0000
UP28	{P1,P2,B7,AP8,S16,AP10,P4}	0.6269	0.0050
UP29	{P1,P20,B12,AP3}	0.6254	0.0000
UP30	{P1,P3,B19,AP4,S2,AP9}	0.6235	0.0556
UP31	{P1,P8,B15,AP2,S13,AP16,P17}	0.6229	0.0235
UP32	{P1,P7,B6,AP19,S3,AP7,P19}	0.6216	0.0000
UP33	{P1,P16,B11,AP3,S17,AP11,P8,S3}	0.605	0.0000
UP34	{P1,P12,B4,AP12,S9,AP20,P7,S5,S18}	0.6014	0.0000
UP35	{P1,P15,B2,AP8,S17,AP11}	0.5945	0.1718
UP36	{P1,P10,B10,AP18,S13,AP16}	0.5831	0.0269
UP37	{P1,P9,B18,AP13,S9,AP20,P20}	0.5565	0.0000
UP38	{P1,P3,B9,AP10,S19,AP14}	0.5363	0.1363
UP39	{P1,P18,B8,AP5,S2,AP9,P3,S2}	0.5338	0.1718
UP40	{P1,P17,B15,AP7,B1,AP4,P2}	0.5059	0.0000

3.3.4 Features Refining

Features refining component takes input from the smart learner. Refining features is a key step before analysing machine-learning performance because some irrelevant features can produce inaccurate classifications. Refinement of extracted features directly effects the time and space consumption for features matching and storing. This component is dependent on Feature extraction and smart learner components. With the feature extraction phase, a huge number of features are obtained. However, some of the extracted features are used only by a few of Android applications such as RSSI_CHANGED, PASSPOINT_ICON, which are not enough to be considered for further analysis. On the other hand, some of the other features are used widely by botware and benign applications almost in the same amount such as INTERNET, READ_PHONE_STATE, which can be hardly considered to distinguish benign and botware applications. Moreover, these features consist of a high-dimensional feature vector, which may cause very complicated computation and cause low efficiency in building detection approach. Based on these generic steps and the components of the proposed approach, features refining algorithm are presented to serve the end users as algorithm for features refining process. Therefore, features refining algorithm is mandatory to be used before machine learning modelling.

Furthermore, the detail of selected features given in Appendix B, include a huge number of features that are extracted from the AndroidManifest.xml and DEX classes of Android applications. These features does not mean that they are significantly useful for detection of Android botnets classification (Yerima *et al.*, 2014a). In this case, there are some features which exist mostly in all benign and botware applications. In order to reduce these features set, we applied features refining method. Features refining is the second last and the most important process in the proposed approach for Android botnet detection. It is a way to enhance the performance of selecting preferred set of features. In this process all the features are assigned a real-valued weight by using the WEKA tool for example INTERNET=0.98, RECEIVE_BOOT_COMPLE-TED=0.897 SEND_SMS=0.868 (Hall *et al.*, 2009). These weights have range from 0 to 1 and these features are initially quantized with fixed precision as shown in the above example. The one whose values is approaching to "1" shows the importance of feature for botnet detection, while the one approaching to "0" is less important.

For this purpose, a filter approach is used. It performs the features selection in this method, by considering its fast execution and generalization. In this approach, Information Gain algorithms were applied to the malicious dataset (Shabtai *et al.*, 2011). Information Gain algorithm is the most used feature selection method in malware detection techniques (Ahmed *et al.*, 2009). While all of these methods followed the feature ranking approach on the basis of specific metrics, the value is computed and returns the score for each feature individually. There was a problem in selection of correct number of features for appropriate classification of botware and benign from the given feature selection algorithm. In order to avoid any partiality in the feature selection an arbitrary number of features in the information gain technique are used.

Information gain measures the amount of information in bits about the class prediction provided the only information available is the presence of a feature and the corresponding class distribution. Let $x = \{Uf_1, Uf_2, Uf_3, \dots \dots \dots Uf_n$ be the used feature set of each application, n represent the total number of used features. Here the information gain value depends on the unique pattern of used features where $UP_{1,}UP_{2,}UP_{3,} \dots \dots, UP_n$ represent the unique pattern used by each application. The class value is needed to calculate the information gain value, for this purpose we consider the C be a random variable to denote the class as botware or benign such as $C \epsilon \{\text{Botware, Benign}\}$. The information gain values that are corresponding to the class label C, are calculated for unique patterns by using the Equation 3.6 and Equation 3.7. The expected information is calculated by using the Equation 3.6 while the entropy is calculated by using the Equation 3.7. Let U be a set with u data samples with m distinct class labels. The training set contains u_i sample of class *i*.

$$Info(u_1, u_2, u_3, \dots \dots \dots u_n) = -\sum_{i=0}^{m} \frac{u_i}{u} log_2(\frac{u_i}{u}) \qquad 3.6$$

Where $\frac{u_i}{u}$ is the probability that a random sample belongs to the class u_i and is estimated by $|U_{i,u}|/|U|$. In order to identify the label of class it need the information which is the average amount of $Info(u_1, u_2, u_3, \dots \dots \dots u_n)$, it is just the average and it is also known as entropy of $(u_1, u_2, u_3, \dots \dots \dots u_n)$. Now let unique pattern *UP* has *v* distinct values $\{UP_1, UP_2, UP_3, \dots \dots \dots UP_v\}$ which can divide the training set into *v* subsets $\{P_1, P_2, P_3, \dots \dots \dots P_v\}$. Where S_i is the subset which has the value of UP_i for *UP*. Let P_j contains $P_{i,j}$ sample of class *i*. The entropy of the unique pattern UP is find by using the Equation 3.5.

$$E(UP) = \sum_{j=1}^{v} \frac{UP_{1,j}, UP_{2,j}, \dots UP_{m,j},}{UP} \times Info(u_{1,j}, u_{2,j}, u_{3,j}, \dots \dots \dots u_{n,j}) \qquad 3.7$$

For the computing of information gain value, WEKA is used (Hall et al., 2009). The IG is obtained by using the Equation 3.8. It is the defined as the difference between the original information requirements and the new requirements.

$$IG(UP) = Info(u_1, u_2, u_3, \dots \dots \dots u_n) - E(UP) \qquad 3.8$$

Feature refining algorithm as shown in Figure 3.9 gives more details about the botware and benign features refining from the original feature set. The input has five parameters, *Fn, αBt, βBt, αBn,* and *βBn.* Where Fs represents the complete original features set, extracted from the dataset using androguard tool. However, the remaining parameters are the thresholds for botware and benign features. *Fn'* is the output containing susceptible features to botware. In general, *Fn'* will always be smaller in size from the *Fs* such as (*Fn'*<*Fn*). In features refining algorithm, and for each feature *fi*, in *Fn*, *NBn*, and

NBt are the numbers of benign and botware applications features which contains *fi* respectively. Furthermore, *rBn*, and *rBt* are the percentage of used features of *NBn* and *NBt* for benign and botware applications.

Table 3.5 Symbol with Description

Symbol	Description
NBt	The total number of Botware applications features
NBn	The total number of Benign Applications features
Fn	It represents the complete original features set
Fn'	The set of susceptible features to botware
αBt	The threshold for botware applications (0.5<*αBt*<1)
βBt	The threshold for botware applications (0<*βBt*<1)
αBn	The threshold for benign applications (0.5<*αBn*<1)
βBn	The threshold for benign applications (0<*βBn* <1)
rBn	The percentage of used features of Benign applications
rBt	The percentage of used features of Botware applications
UP	It represents the complete used pattern
UP'	The set of susceptible used pattern to botware

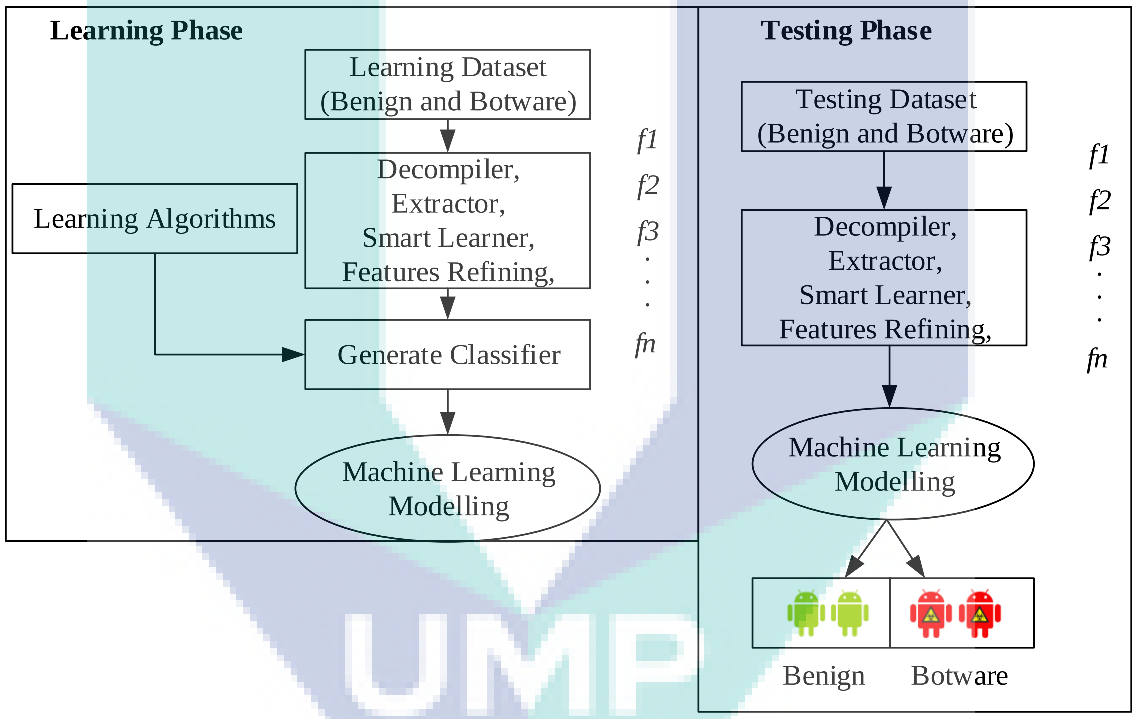
In the features refining algorithm, *Bt* and *Bn* are the total numbers of botware and benign applications. However, *αbt* and *βBt* are the thresholds where $0.5 \leq \alpha Bt \leq 1$ and $0 \leq \beta Bt \leq 1$ for botware applications, and $0.5 \leq \alpha Bn \leq 1$ and $0 \leq \beta Bn \leq 1$ for benign applications. The two conditions are examined which are $rBt \geq \alpha Bt$ and $NBt/Bt \geq \beta Bt$. In the first condition, it implied that a feature is used more frequently in botware applications than benign applications while the second condition suggested that the time of occurrence of a feature in all botware exceeds threshold *βBt*. The same procedure is applied for benign features as well. The pattern in *Fn′* are collected using two different ways. The botware pattern which are frequently used are collected using code from line fifteen to nineteen, while the pattern which are frequently used by benign are collected by the rest of the code.

```
Input: Extracted Features,
Output: UP′,
1:  Calculate the Occurrence frequency of each feature in  and
2:      $Fn = \frac{\sum_{i=1}^{n} Fi}{NBt}$        where n the nth feature in $DBt$
3:      $Fn = \frac{\sum_{i=1}^{n} Fi}{NBn}$        where n the nth feature in $DBn$
4:      Calculate support value
        Support (Pi,Ai,Bi,Si,APi) =  (Number of Applications that contains Pi, Ai, Bi,
                                     Si, APi in a $DBt$) / (Total number of Applications in $DBt$)
5:  UP ← Support
6:  Set a Threshold value equal to 0.5
7:  If UP > = Threshold value
8:          UP ← Botware
9   Else
10:         UP ← Benign
11: End if
12:     for $i$ → 1 to UP.$size$() do
13:             $NBt$ ← $FeaturecountInBotware(fi)$;
14:             $NBn$ ← $FeaturecountInBenign(fi)$;
15:             $rBt$ ← $NBt/(NBt + NBn)$;
16:             if $rBt \geq \alpha Bt \&\& NBt/Bt \geq \beta Bt$ then
17:                     UP′← $fi$;
18:                     else UP′↚ $fi$;
19:                 end if
20:             $rBn$ ← $NBn/(NBt + NBn)$;
21:             if $rBn \geq \alpha Bn \&\& NBn/Bn \geq \beta Bn$ then
22:                     UP′← $fi$;
23:                     else UP′↚ $fi$;
24:                 end if
25:     end for
26:     return(UP′);
```

Figure 3.9   Feature Refining Algorithm

3.3.5 Machine Learning Tools

In this approach, the final component is machine learning modelling. The machine learning algorithm are classified in to three main categories such as Supervised, Unsupervised, and semi supervised. Supervised learning algorithm are utilized to deal with labeled dataset. However, unsupervised learning algorithm is utilized when the dataset is unlabeled. Moreover, the semi supervised machine algorithm is the mixture of both supervised and unsupervised learning algorithms. It deals with small amount of labeled dataset, and on the base of this it assigned labels to unlabeled dataset. Since the dataset used in this research study is labeled and has two target classes including Benign and Botware, supervised learning algorithms are preferred for classification. The choice of an appropriate selection technique depends on the nature of the Android application. In this study, selecting classifier is based on the performance, number of classes and ranking criteria of features. This study explored WEKA, which is a data mining software written in Java (Hall *et al.*, 2009; Smith & Frank, 2016). Since the prototype of final approach is implemented in Java and there is need to use the generated component for classification in the prototype, then, WEKA was decided for and used for this step. Figure 3.10 shows the block diagram of machine learning classifiers.



Learning Phase
Learning Dataset (Benign and Botware)
Decompiler, Extractor, Smart Learner, Features Refining,
Learning Algorithms
Generate Classifier
Machine Learning Modelling
f1
f2
f3
fn
Testing Phase
Testing Dataset (Benign and Botware)
Decompiler, Extractor, Smart Learner, Features Refining,
Machine Learning Modelling
f1
f2
f3
fn
Benign
Botware

Figure 3.10 Life Cycle of Machine Learning Modelling

During the classification phase, proper machine learning algorithms were selected to recognize the botware applications with an adequate accuracy. It is an important task to choose an appropriate classifier to generate a reliable detection approach, which ultimately demonstrates the accuracy of the detection approach in all. Therefore, in order to choose a proper machine learning algorithm, the following requirements are considered: (1) diverse feature domain: the total number of static features are considered from a multiple domain; (2) sparse feature set: the supreme features set are finally picked for the proposed approach evaluation; (3) scalability: the system should be scalable enough to accommodate the future requirements of users; and (4) performance: algorithm performance in order of testing and training should be minimal to provide a prompt response to the user. Given the abovementioned consideration, Random Forest (RF), Naïve Bayes (NB), Support Vector Machine (SVM), Simple logistic Regression (SLR), and J-48 are selected as the classification algorithms to establish and test the proposed approach. These selected algorithms are discussed in Section 2.9 with more detail. Once the features are refined, the next stage is to train the machine learning classifier. The dataset is split using *k*-fold cross validation technique. This technique divides the input dataset into *k* times. One subset is used for shaping sample dataset, called test set, while the *k-1* of subsets forming the joint training.

3.4 Conclusion

In this Chapter, the methodology for proposed approach is discussed. The proposed approach is divided into five main components namely: decompiler, features extractor, smart learner, features refining and machine learning modelling. The first component decompiles the applications for AndroidManifest.xml and DEX file by using the APK tool. In the second component, reverse engineering is applied to extract all the static features (permissions, activities, broadcast receivers, services, and API calls) from Android applications by using the Androguard tool. However, the first four features are extracted from the manifest.xml file while the API calls is extracted from the DEX classes. The reasons for choosing these features are described in sections 3.3 in more details. These features are extracted from the applications in selected dataset from both categories (botware and benign). In the smart learner component, applying the Apriori algorithm in WEKA tool. It allotted a specific frequency to each feature in permissions, activities, broadcast receivers, services, and API Calls according to their usage. Using a proper frequency analysis, some of the unique patterns are selected in each category that can cause a botnet attack. However, the features refining phase is used to refine the unique patterns on the base of their frequencies and botnet susceptible characteristics. Furthermore, machine learning algorithms (support vector machine, J48, random forest, simple logistic regression, and Naïve Bayes) are applied. In conclusion, the botnet phenomenon has migrated progressively from the previous generation personal computer based on the new emerging computation intensive mobile platform. Therefore, practical devices through which users are made aware of the consequences of unknowingly installing an application with botnet intention should be designed. Botnet attacks do not only affect the overall performance of a device but also forces a mobile device to help unintentionally the spreading of the cybercriminal attack.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Overview

In this chapter, the performance evaluation approaches are described to evaluate the proposed approach (modified). For this purpose, the performance difference is analyzed between unmodified (without features refining component) and modified (with features refining component) by considering the performance parameters such as True Positive Rate (TPR), False Positive Rate (FPR), Precision, F-Measure, and Accuracy was carried out. The main motive of this chapter is to discuss and analyze the experimental setup, tools, evaluation parameters, and to analyze the performance of the proposed approach. Furthermore, the remaining chapter is divided into nine sub-sections. Section 4.2 illustrates the experimental tools, Section 4.3 discussed the experimental setup and results. The evaluation process of the proposed approach is given in Section 4.4 while Section 4.5 details the evaluation methods. Furthermore, the evaluation parameters are explained in Section 4.6. Section 4.7 contains the evaluation of machine learning classifiers based on individual features, while the Section 4.8 depicts the evaluation of machine learning classifiers based on unique patterns. Performance analysis is described in Section 4.9 with the chapter summary in Section 4.10.

4.2 Experimental Tools

There are various types of tools used in mobile malware detection and analysis. This section described the tools that are used to perform experiments on mobile malware detection. These tools are used to analyse the APK files and help to improve the analysis process more effectively and accurately. Static analysis tools have the capability to inspect the different components of Android applications. The list of experimental tools is given in the following sections.

4.2.1 Androguard

Androguard provides support for decompiling the APK file and dissecting the APK file into its original components (Desnos, 2011a). It is an open source tool written in python are employed for reverse engineering of Android APK files and statically analysing the Dalvik bytecode. This tool detects the botware behaviour category in an APK file by searching the pre-defined method calls in the dissembled bytecode. However, the requirements for this study is different from the tool, thus, it is modified according to the need of this study.

4.2.2 Android Application Package (APK) Tool

The ApkTool is a reverse engineering tool that is used for reverse engineer the Android Applications. It has the capability to decode and rebuild the Android applications almost to its original condition after performing some modification. The researchers use this tool in order to add some features, localization and analysing the Android applications.

4.2.3 Machine Learning Tools

Machine Learning tools has the ability to learn from the existing dataset and implement prediction or decision on the new samples. The performance of the system can be improved with high impact by implementation of machine learning tools. Supervised and unsupervised are the two main types of machine learning tools. Furthermore, it has the skills to apply complex mathematical equations automatically to solve the complex problems. In this study WEKA is used for experiments. At the same time, it is also used for the analysis tasks. WEKA is an open source tool deployed for using different machine learning algorithms (Hall *et al.*, 2009). It provides the functionality of pre-processing, clustering, classification, regression and visualization. Likewise, it provides an interface through software package with GUI as well as Java APIs. Figure 4.1 shows the WEKA graphical user interface while, this tool is implemented in Java. According to the scope of the current study, this tool is used for the classification of data and the Java APIs for implementing the approach. There are several classifiers used in this study so as to compare the performance of the proposed approach. The detail will be discussed in the following sections.

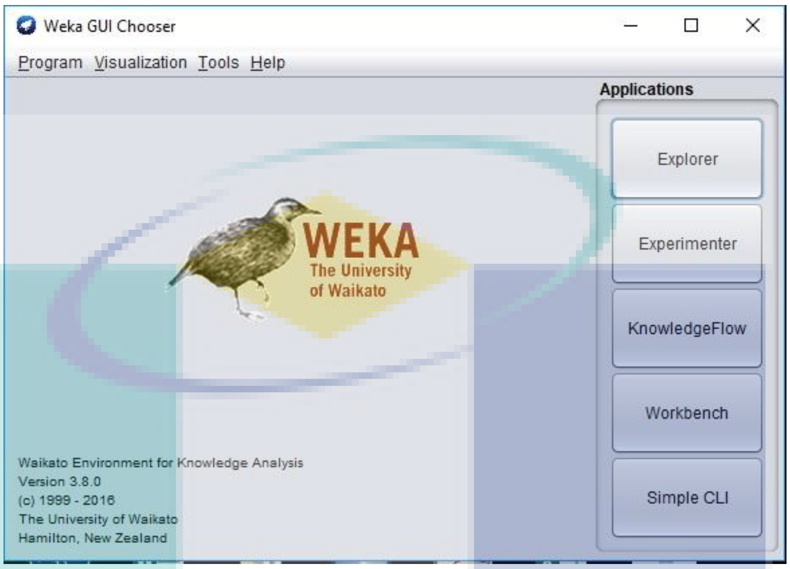


Figure 4.1 WEKA Graphical User Interface

4.2.4 Online Analysis Tools

In order to check the Android applications for botware and benign, VirusTotal is used. As long as, its service incorporates a large selection of anti-virus scanners, which uses different strategies for botnet detection. In this study different number of scanners that detect the selected samples as botnet are identified. It can return the botnet detection results of about 59 varies types of antivirus with the latest updated signatures.

4.3 Experimental Setup and Results

This section presents and discusses the experimental setup and obtained results. Furthermore, this section also discusses about the dataset used in the experiments. In order to evaluate the proposed approach, an experimental setup was created with Ubuntu OS 64bit. For this purpose, Intel(R) Core (TM) i7-7700 CPU with 3.60GHz is used. This system has 16GB of RAM and 1TB of secondary memory. Figure 4.2 depicts the experimental setup.

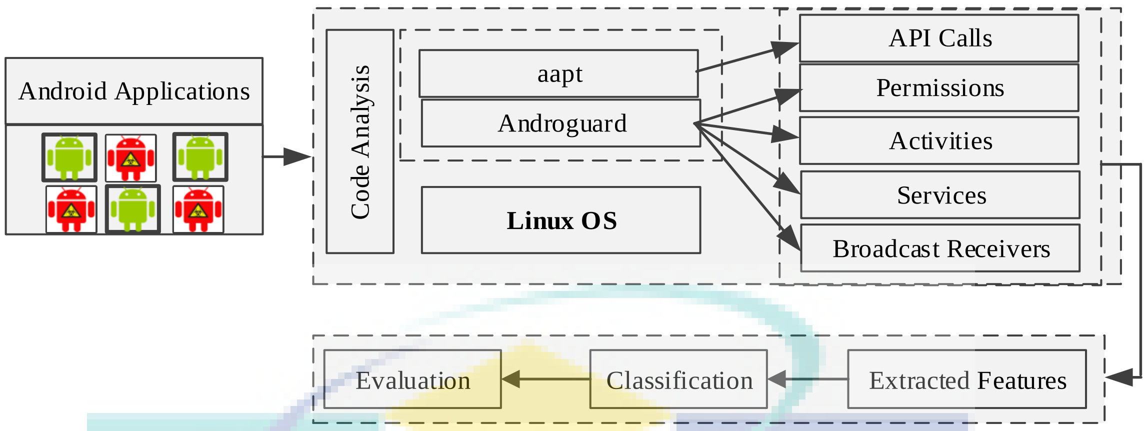


Figure 4.2 Experimental Setup

4.3.1 Used Datasets

The analysis task is achieved, the samples are downloaded from the online repositories, including Google Play Store, Contagio malware repository and the well-known Drebin dataset (Arp *et al.*, 2014; Parkour, 2011; PlayStore, 2017). The Drebin dataset is used as a mobile application repository for this entire study for the purpose of validation of results. At the time of writing this thesis, Drebin is the largest publicly available dataset used by numerous educational institutions. For this study, 7035 mobile applications are initially selected as samples. From this total, 3535 samples are selected from botware category and 3500 samples are selected from benign. The dataset employed for evaluation is described in Table 4.1.

Table 4.1 Botware and Benign Dataset Used

Sample	**Apps**	**Repository**	**Observation**	**Category**	**Family**
Botware	3535	Contagio / Drebin	Code/ Runtime	HTTP/SMS based	Anserverbot,Bmaster, DroidDream, Geinimi, MisoSMS, Nickyspy, NotCompatible, PJapps, Pletor, Rootsmart, Snadroid, Tigerbot,Wroba, Zitmo
Benign	3500	Google Play Store	Code/ Runtime	Games, Wallpapers, Entertainment, GPS, Web Browser, books,	N/A

Botware samples used in this study were obtained from the Android botnet dataset provided by Information Security Centre of Excellence (ISCX) and The Drebin Dataset

(Arp *et al.*, 2014; Kadir *et al.*, 2015), while the benign samples were selected from the Drebin (Arp *et al.*, 2014). Furthermore, ISCX contains 1929 Android botnet applications divided into fourteen different families according to their behaviour as shown in Table 4.2 while the benign applications are listed in Table 4.3.

Table 4.2 Families of Android Botnets

Source	**Family**	**C&C**	**Year**	**Total Sample**	**Propagation and Attack Types**
ISCX	AnserverBot	HTTP	2011	244	Backdoor, Infected SMS, Social Engineering
	Bmaster	HTTP	2012	6	Data Theft, Exploit Technique, Repackaged Application
	DroidDream	HTTP	2011	363	Data theft, Drive-by Download, Exploit Technique, Repackaged Application, Trojanized Applications
	Geinimi	HTTP	2010	264	Data theft, Drive-by Download, Repackaged Application
	MisoSMS	Email	2013	100	Data Theft, Exploit Technique, Trojanized Application
	NickySpy	SMS	2011	199	Data Theft, Repackaged Application
	NotCompatible	HTTP	2014	76	Drive-by Download, Exploit Technique
	PJapps	HTTP	2011	244	Repackaged Application, Trojanized Application
	Pletor	SMS/HTTP	2014	85	Ransomware, Trojanized Application
	RootSmart	HTTP	2012	28	Data Theft, Exploit Technique, Repackaged Application
	Sandroid	SMS	2014	44	Mobile Banking Attack, Ransomware, Trojanized Application
	TigerBot	SMS	2012	96	Backdoor, Data Theft, Trojanized Application
	Wroba	SMS/HTTP	2014	100	Infected SMS, Mobile Banking Attack, Trojanized Application
	Zitmo	SMS	2010	80	Infected SMS, Mobile Banking Attack, Repackaged Application, Social Engineering
The Drebin Dataset	---	SMS/HTTP	2016	1606	

Table 4.3 Benign Applications

Samples	C&C	Year	Total Sample	# of Selected Samples	Source
Benign	---	N/A	14865	3500	Google Playstore

4.3.2 Pre-Processing

The selected samples from the datasets given in Section 4.3.1 are initially checked with VirusTotal (Virustotal, 2017). VirusTotal provides a platform for checking the applications online. The dataset chosen for these experiments shows that it contains 90% of the malware that existed in August 2016. By random selection the applications are obtained using a Monte Carlo sampling method. With this different version for the same application is avoided while there are different types of applications such as native, web, and widgets (Sanz et al., 2013). Different types of application have different types of features to construct a dataset; hence, the application is randomly selected without keeping the distinction in their features.

4.3.3 Results

This section describes the experimental results that are performed by proposed approach on Botware and benign applications. The details of the static features (permissions, activities, broadcast receivers, services, and API Calls), which are used in these tests are provided in Section 2.4 and Section 2.5. Table 4.4 listed the comparison of permissions features generated from botware and benign applications while the Figure 4.3 shows the frequencies of the aforementioned features. The ranges of botware for various features are from 18.57 % to 97.86 % and for benign applications are from 1.43 % to 51.43 %. The average for botware applications is 54.25 % and for benign applications is 11.32 %. This enormous difference shows that botware application requests a greater number of permissions features as compared to the benign applications. For instance, one of the permission features of botware and benign applications is INTERNET permission. In addition, the requests generated by botware applications are 97.86% while 51.43% for the benign applications. Therefore, the smartphones users should be aware of the botware's susceptible permissions features during the installation of any Android applications.

Table 4.4 Comparison of Top 20 Requested Permissions by Botware & Benign (%)

Selected Features	**Botware**	**Benign**
INTERNET	97.86	51.43
READ_PHONE_STATE	95.71	31.43
READ_CONTACTS	81.43	23.57
SEND_SMS	80.71	15.00
READ_SMS	77.14	7.86
RECEIVE_SMS	71.43	5.71
CALL_PHONE	66.43	8.57
WRITE_SMS	65.00	6.43
WRITE_SETTINGS	62.14	9.29
WRITE_CONTACTS	58.57	2.86
CHANGE_WIFI_STATE	50.71	17.86
ACCESS_FINE_LOCATION	47.86	4.29
SYSTEM_ALERT_WINDOW	47.14	10.00
GET TASKS	40.00	3.57
DISABLE_KEYGUARD	31.43	11.43
ACCESS_COARSE_LOCATION	30.71	5.71
CAMERA	22.14	1.43
BLUETOOTH	20.00	2.86
PROCESS OUTGOING CALLS	20.00	2.86
RECORD_AUDIO	18.57	4.29
Minimum	**18.57**	**1.43**
Maximum	**97.86**	**51.43**
Mean	**54.25**	**11.32**

The list of permissions and their frequency are given in Figure 4.3. It shows the standard permissions used by botware and benign applications. In this figure, the X-axis shows the number of used permission features by each application while the Y-axis shows the list of permissions. The figure clearly describes that the botware applications used more features as compared to benign applications. Since the INTERNET permission is used by botware to establish a remote connection with command and control (C&C) server as shown in the Figure 4.3. Therefore, the most used permission is INTERNET because connection with C&C server is constructed to observe the state of the target device and network as well as to read personal credentials.

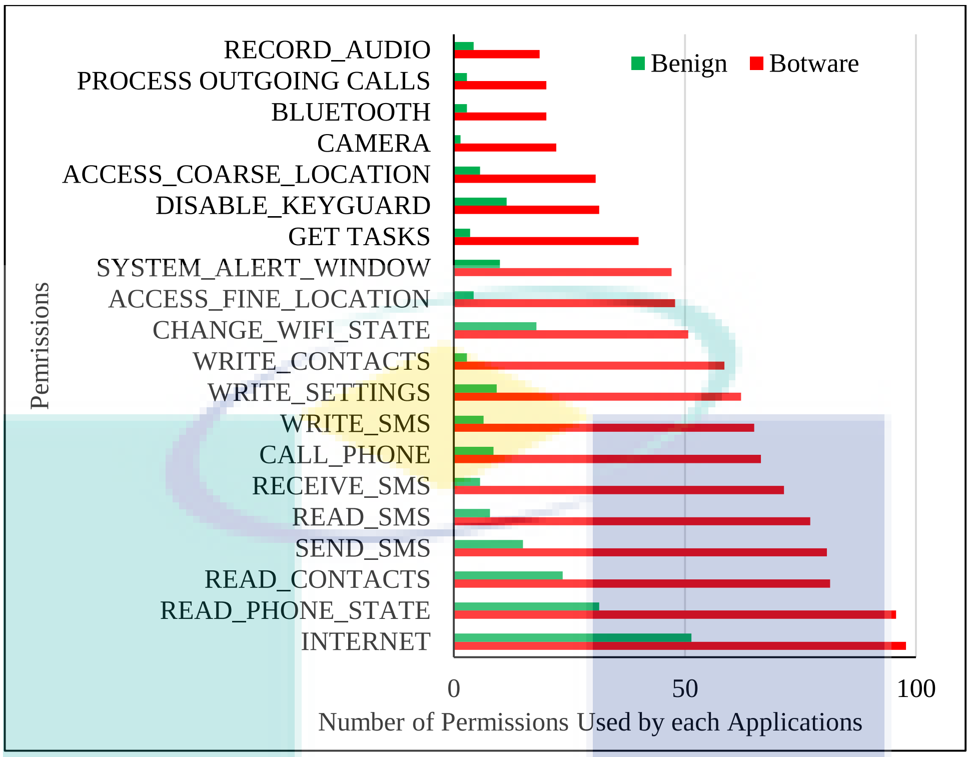


Figure 4.3 Permissions Frequency Analysis Comparison between Botware and Benign

Table 4.5 presents the number of used activities features by botware and benign applications. The result shows that the botware and benign application used a set of 793 activities in total. However, the ranges of various botware applications for activities features are from 12.14 % to 92.14 % and for benign applications are from 0.71 % to 6.43 %. The average for botware applications is 19.43 % and for benign applications is 2.66 %. This huge difference shows that botware requests more number of activities features as compared to the benign applications. For instance, one of the activity feature of botware and benign applications is "about" activity. In addition, the requests generated by botware applications are 92.14 % while 6.43 % for the benign applications. It shows that the botware applications usually generate more requests as compared to benign applications.

Table 4.5 Comparison of Top 20 Activities Used by Botware and Benign (%)

Selected Features	**Botware**	**Benign**
About	92.14	6.43
About App	27.86	2.14
About Spanish Trainer	18.57	1.43
Accept Challenge	18.57	1.43
Acheter Version Payante	18.57	2.14

Table 4.5 Continued

Selected Features	Botware	Benign
Achievement	18.57	0.71
Achievement Header	16.43	1.43
Achievement List	16.43	2.14
Achievements Screen	14.29	1.43
Acts View	14.29	2.86
AdActivity	13.57	5.00
Add Entry	13.57	4.29
Add Radar Form	12.86	3.57
Add Review	12.86	3.57
Add to Contact	12.14	2.86
Add Your Pic	12.14	2.86
AddByHand	12.14	2.14
AddByWeb	12.14	2.14
AdMob	12.14	2.14
Minimum	**12.14**	**0.71**
Maximum	**92.14**	**6.43**
Mean	**19.44**	**2.67**

Figure 4.4 shows the standard activities that are used by botware and benign applications. From the Figure, it is clearly shown that the botware use more activities features as compared to the benign applications. Furthermore, the most prominent activities used by botware and benign applications are about, about App, about Spanish trainer, accept challenge, acheter version payante and much more.

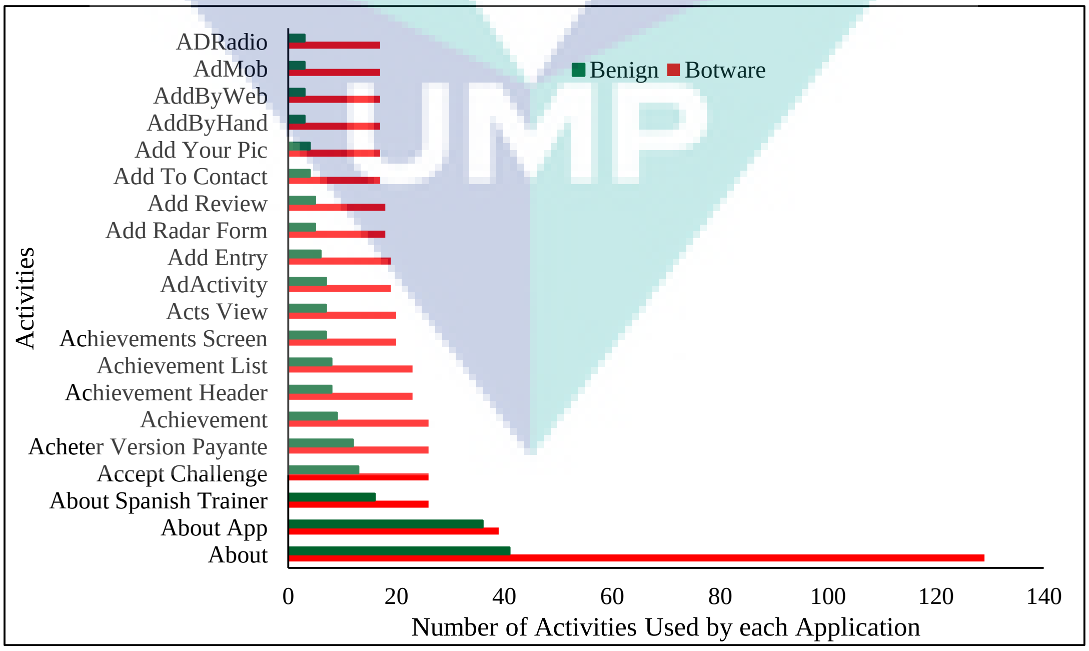


Figure 4.4 Activities Features Frequency Analysis

Table 4.6 illustrates the number of used broadcast receivers by botware and benign applications. These results illustrate that botware and benign application are using a set of 347 broadcast receivers in total. The ranges of botware for various features are from 3.57 % to 12.85 % and for benign applications are from 0.71 % to 2.85 %. Similarly, the average for botware applications is 8.60 % and for benign applications is 0.92 %. This huge difference also shows that botware used more number of broadcast receiver features as compared to the benign applications. For instance, one of the broadcast receiver features of botware and benign applications is "com.clientsoftware.MessageReceiver" feature as shown in the Table 5.6. The requests by this feature generated 12.85 % by botware applications and 2.85 % for the benign applications. It shows that the botware applications usually generate more requests as compared to benign applications.

Table 4.6 Comparison of Top 20 Broadcast Receivers Used by Botware and Benign (%)

Selected Broadcast Receivers Features	**Botware**	**Benign**
com.clientsoftware.MessageReceiver	12.86	2.86
com.clientsoftware.MyDeviceAdminReceiver	12.86	1.86
com.clientsoftware.SDCardServiceStarter	12.86	1.71
com.clientsoftware.ServiceStarter	12.86	2.71
com.phone.callcorexy.xy.LScreen	12.14	1.31
com.phone.callcorexy.xy.SReceiver	12.14	2.32
com.phone.callcorexy.xy.StartupReceiver	12.14	0.71
com.bwx.bequick.flashlight.LedFlashlightReceiver	11.43	0.71
com.bwx.bequick.receivers.StatusBarIntegrationReceiver	11.43	0.71
com.clientsoftware.InternetReceiver	11.43	0.71
com.google.android.mms.BootReceiver	6.43	0.71
com.google.android.mms.LiveReceiver	6.43	0.71
com.google.android.apps.analytics.AnalyticsReceiver	5.71	0.71
com.google.android.mms.WakeLockReceiver	5.71	0.71
com.a.a.A	5.00	0.71
com.a.a.DeAdminReciver	5.00	0.71
com.admv3.listener.OnBootReceiver	4.29	0.71
com.devy.entry.LSecScreen	4.29	0.71
com.a.a.SystemR	3.57	0.71
com.android.XWLauncher.InstallShortcutReceiver	3.57	0.71
Minimum	**3.57**	**0.71**
Maximum	**12.86**	**2.86**
Mean	**8.61**	**0.93**

The broadcast receivers which are commonly used by botware and benign applications are listed below in Figure 4.5. The examples include: com.clientsoftware.MessageReceiver,com.clientsoftware.MyDeviceAdminReceiver, com.clientsoftware.SDCardServiceSta-rter,com.clientsoftware.ServiceStar-ter, com.phone.callcorexy.xy.LScreen, com.phone-.callcorexy.xy.SReceiver, and com.ph-one.callcorexy.xy.StartupReceiver are the common broadcast receivers used by botware and benign with a ratio of 18:4, 18:4, 18:1, 18:1, and the remaining are 17:1 respectively.

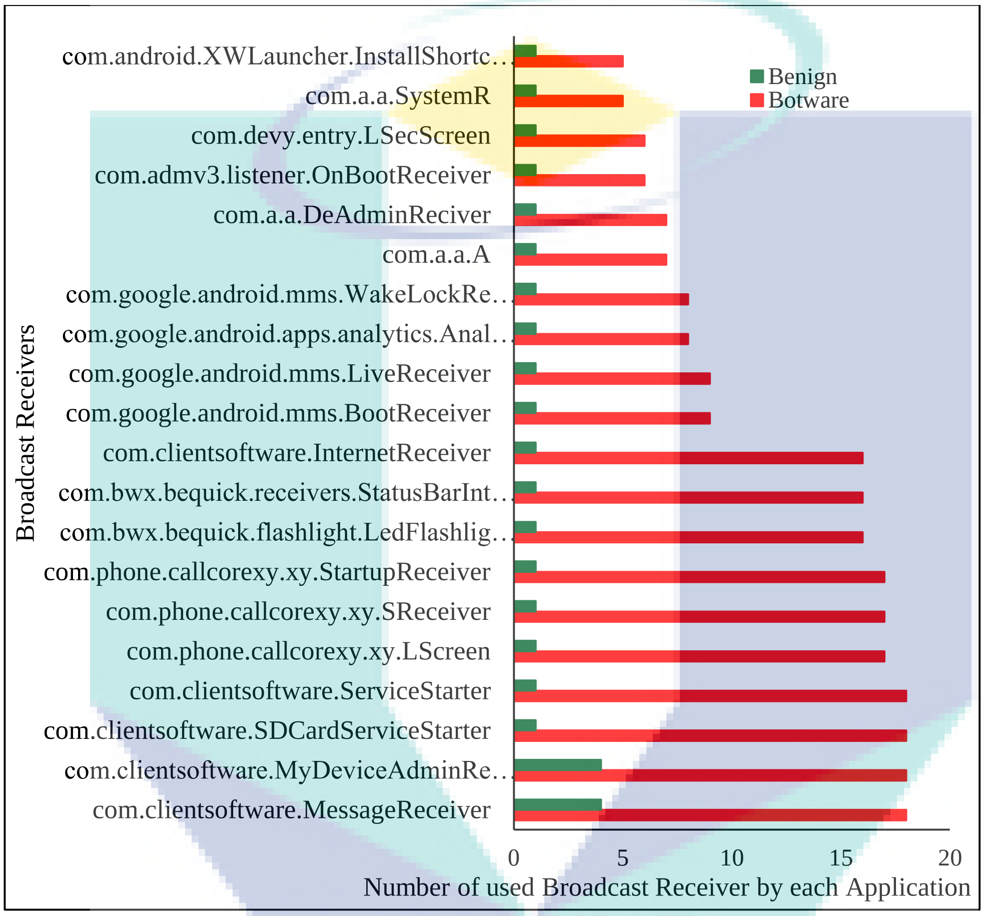


Figure 4.5 Broadcast Receivers Frequency Analysis

Table 4.7 illustrates the number of used services by botware and benign applications. However, the obtained result shows that botware and benign applications are using a set of 292 services in total. The ranges of botware for various features are from 4.28 % to 17.85 % and for benign applications are from 0.00 % to 1.42 %. The average for botware applications is 10.28 % and for benign applications is 0.71 %. As reported earlier, this huge difference shows that botware requests more number of services features as compared to the benign applications. For instance, one of the service feature of botware and benign applications is “FourthAService” service as shown in the Table 4.7. As an illustration, the requests generated by botware and benign applications are 17.85 % and 0.71 % respectively. It shows that the botware applications usually generate more requests as compared to benign applications. Therefore, the smartphones users should be aware of the botware's susceptible services features during the installation of any Android applications.

Table 4.7 Comparison of Top 20 Services Used by Botware and Benign (%)

Selected Services Features	**Botware**	**Benign**
FourthAService	17.86	0.71
SecondAService	17.86	1.43
ThirdAService	17.86	0.71
com.baidu.location.f	12.86	0.00
com.phone.callcorexy.CallLogger	12.86	0.71
com.phone.callcorexy.xy.CRSService	12.86	1.43
com.phone.callcorexy.xy.SService	12.86	0.71
com.Security.Update.SecurityUpdateService	12.86	0.00
com.Rockstargames.CheckService	11.43	0.71
com.Rockstargames.DecryptService	11.43	1.43
com.Rockstargames.MainService	11.43	0.71
com.android.main.MainService	9.29	0.00
com.admv.service.AdvService	7.86	0.71
com.admv.service.MainService	7.86	1.43
com.google.android.mms.MainService	6.43	0.71
com.umeng.common.net.DownloadingService	5.00	0.00
com.android.security.SecurityService	4.29	0.71
com.nl.MyService	4.29	1.43
com.un.service.autoRunService	4.29	0.71
com.un.service.CallService	4.29	0.00
Minimum	**4.29**	**0.00**
Maximum	**17.86**	**1.43**
Mean	**10.29**	**0.71**

Figure 4.6 shows the frequently used services by benign and botware applications. Most of the Android botware applications use more services as compared to benign applications. In the statistics, it is clear that FourthAService, SecondAService, ThirdAService, com.baidu.location.f, com.phone.callcorexy.Call-Logger, com.phone.ca-llcorexy.xy.CRSService, and com.phone.callcorexy.xy.SService have the ratio of 25:1 for the first three service feature while 18:1 for the next four services respectively.

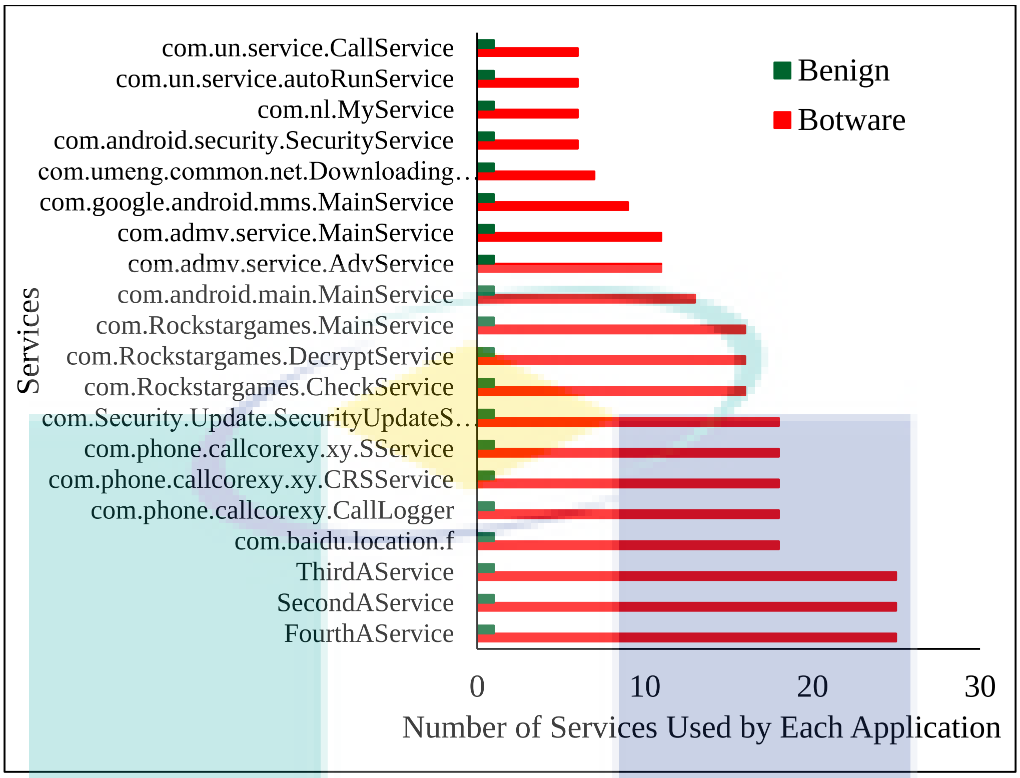


Figure 4.6 Services Frequency Analysis

Table 4.8 illustrates the number of used API calls by botware and benign applications. The result shows that both botware and benign applications use a set of 15 API Calls in total. The ranges of botware for various features are from 5.00 % to 75.71% and for benign applications are from 0.71 % to 10.71 %. The average for botware and benign applications is 30.19 % and 2.57 % respectively. This huge difference as observed for other previous features shows that botware requests a greater number of API calls features as compared to the benign applications. For instance, one of the API call features of botware and benign applications is “connect” as shown in the Table 4.8. The requests generated by botware applications are 75.71 % while the benign applications generated 10.71%. Thus, this shows that the botware applications usually generate more requests as compared to benign applications.

Table 4.8 Comparison of API Calls Used by Botware and Benign (%)

Selected API_Calls Features	Botware	Benign
connect	75.71	10.71
getDeviceId	57.14	2.14
getSubscriberId	45.00	1.43
getActiveNetworkInfo	36.43	2.14

Table 4.8 Continued

Selected API_Calls Features	Botware	Benign
getLine1Number	34.29	1.43
getNetworkInfo	33.57	1.43
getSimSerialNumber	30.71	1.43
getInputStream	28.57	0.71
sendTextMessage	26.43	1.43
getLastKnownLocation	22.86	2.14
LocationListener	20.71	2.14
requestLocationUpdates	20.71	4.29
getContent	8.57	3.57
getWifiState	7.14	2.14
openFileDescriptor	5.00	1.43
Minimum	**5.00**	**0.71**
Maximum	**75.71**	**10.71**
Mean	**30.19**	**2.57**

The API calls are indeed helpful for differentiating benign and malware applications, hence, the top 10 API calls used in botware and benign applications are reported in Figure 4.7. The results clearly show that botware applications use more API calls than benign applications. This is further expressed by the average where the average number of API calls used by botware and benign applications are 106 and 10 respectively.

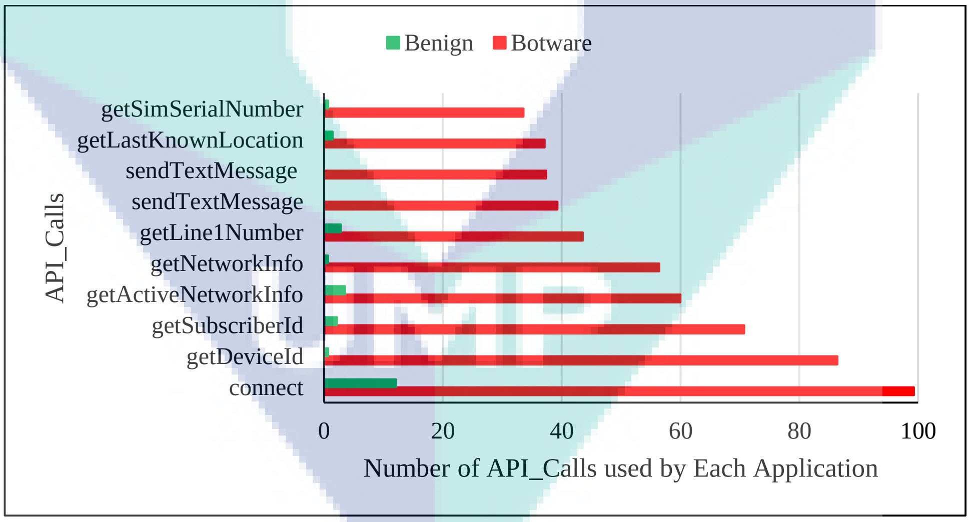


Figure 4.7 API Calls Frequency Analysis

4.4 Evaluation Process

In order to evaluate the performance of the selected classification method, series of experiments are conducted using k-fold cross-validation technique. For this purpose, we selected five different types of machine learning classifiers namely, Random Forest, SVM, J48, SLR, and Naïve Bayes given in Section 2.9 in more detail. Initially, a dataset of 7565 real Android applications are considered for all experiments in this study. Botnet datasets are obtained from the Drebin and ISCX as accounted for in Section 4.1.3. However, the benign applications are obtained from Google Play-store and other third party sites (Excellence, 2016; PlayStore, 2017). Some of the malware samples are collected from the Android Malware Genome Project (Arp *et al.*, 2014; Excellence, 2016). In order to cross-check normality of the selected samples, Virustotal services are used (Virustotal, 2017). After eliminating 530 duplicate samples, 7035 samples are left from both applications with 3535 botware and 3500 benign applications respectively. The obtained results are shown in the following sections. For all experiments, the dataset of real Android applications is considered for Benign and Botware.

In this study, a list of experiments were performed in WEKA experimental tool (Hall *et al.*, 2009). It is a powerful feature of the WEKA workbench and a perfect tool to perform a wide-range machine learning experiment. WEKA has a built-in graphical user interface explorer that is used for experiments on different type's machine-learning algorithms for large datasets. The graphical user interface is robust enough to produce a large number of experimental results needed for evaluation and comparison. Initially, the model is trained on the labeled botware applications. The features of these labeled botware are given in Section 2.4 and Section 2.5. In general, the validation of machine learning classifiers is performed in two ways. The first one is k-fold cross validation while the second one is random sampling validation (Bouckaert & Frank, 2004).

In K-fold cross validation, the dataset is randomly split into K equal-sized subsamples. From the K subsamples, a single subsample is selected as the validation dataset for testing the model, whereas the remaining K-1 subsamples are regarded as the training datasets. The process is repeated K times (thus the term K-fold), with each of the K subsamples used exactly once for validation. The results of the K-fold cross validation are then averaged to generate a single assessment. The advantages of this method over random sampling validation are the use of all instances for both training and testing, and each observation is used only once as a test instance.

However, Random sampling method randomly separates the dataset into training and testing data. In each split, the model is designed using the training data, whereas the predictive accuracy is measured using the testing data. The average results are then obtained from each split. The obvious advantage of this method over K-fold cross validation is that the proportion of split for the training and testing data does not depend on the number of K-folds. Nevertheless, overlapping of validation subsets may occur in this situation, in which some instances may never be used in validation subsamples or some observations may be selected repeatedly. Moreover, the generated results may vary if the analysis is performed with different random splits. Therefore, K-fold cross validation was utilized to validate the accuracy of our classifier model. We used different value for K from 2 to 10 in the cross-validation tests.

4.5 Evaluation Methods

In order to analyse the reliability and validity of the research, several statistical analyses are performed on the collected data through different datasets and executing experiments in a different scenario. A statistical model is used to represent and analyse generated data by an average and a standard deviation. The statistical model always implies dependent and explanatory variable. Computation behind the statistical modelling allows us to show the significance of our research. Each of the statistical methods that are used in this research are presented in the following section.

4.5.1 Descriptive Statistics

The descriptive statistics is used in this research in order to analyse data and to highlight the significance of achievement of the modified approach based on refining features in terms of TPR, FPR, precision, F-Measure and Accuracy of botware detection as compared to unmodified (without refining component) approach. In descriptive statistics, minimum, maximum, mean and the standard deviation measures are determined. The desired descriptive data is acquired based on the collected data and summarized in the graphical and tabular form to accomplish the desired objectives.

4.5.2 Confidence Interval

According to the sample Central Limit theorem, approximately 95 % of the sample means fall within 1.96 standard deviations of the population mean, which showed that the sample is greater than or equal to 30 ($n \geq 30$). Therefore, all the experiments in this research are executed 30 times for the performance evaluation of individual variable to verify that the obtained value belongs to the representative samples. In the data sample, the measurement of the central tendency of each experiment is calculated based on the sample mean (-X). This is carried out so as to discover that sample mean is a better point estimate of the population mean as compared to median or mode. Data sampling includes a range of intervals determined from the specified confidence level, some statistics, and the factor of sampling error; hence the sample mean can differ from the population mean. The level of confidence is the probability that the parameter is truly captured by the confidence range. The most common Confidence Levels (CL) are 90%, 95%, and 99%. Therefore, the interval estimate of each sample is determined in order to signify the goodness of the calculated point estimate. The interval estimate for each sample mean of the primary data is calculated with approximately 95% confidence interval of the sample means within 1.96 standard deviations by using the following equation. Therefore, for reporting the parametric results, the readability and confidence of the results are raised up to 95%. Equation 4.1 is used to calculate the margin of error in the sample with the terms defined.

$$M = \mathrm{Z} * \left(\frac{\sigma}{\sqrt{n}}\right) \qquad 4.1$$

Where, M is the margin of error, Z indicates the value based on the confidence interval percentage, σ is the standard deviation, and n is the number of samples. Again, Equation 4.2 is used to calculate the confidence interval estimates for each sample mean (X) of the primary data with a 95% confidence interval.

$$\mu = \mathrm{X} \pm 1.96\left(\frac{\sigma}{\sqrt{n}}\right) \qquad 4.2$$

Where, σ is used to indicate the standard deviation in the sample values and n shows the size of sample space.

4.5.3 Paired Samples T-Test

In this research, the Paired Samples T-Test was performed so as to ensure that there is a significant difference between the mean values of the identical measurement performed in two different approach namely unmodified (without refine component) and modified (with refine component, the case of our solution) execution modes. In this study, the unmodified approach and the modified approach parametric values are paired data of the same workload into two different execution modes. This was used to ensure that the execution modes of the unmodified (without refine component) and modified (with refine component) approach have a significant impact on the TPR, FPR, precision, F-measure and accuracy or not. In other words, conclusion was drawn with the help of the generated results from the Paired Sample T-Test that the bearable TPR, FPR, precision, F-measure, and accuracy in the unmodified (without refine component) and modified (with refine component) botware detection approach have a significant difference.

4.6 Evaluation Parameters

In this section, the evaluating process of the proposed botnet detection approach was described. This is achieved by describing the criteria through which evaluation of the effectiveness of our approach is performed. The performance of the proposed approach has been evaluated using five different matrices, namely True Positive Rate (TPR), False Positive Rate (FPR), Precision, F-measure, and Accuracy and compared to the existing state-of-the-art detection techniques. Table 4.9 shows the evaluation parameters with description and their possible formulas.

Table 4.9 Performance Evaluation Parameters of the Proposed Approach

Parameters	**Description**	**Formula (if any)**
True Positive Rate (TPR)	When it is actually Botware, how often does it predict as Botware	$TPR = TP/(TP + FN)$
False Positive Rate (FPR)	When it is actually Botware, how does it often predict as Benign	$FPR = FP/(FP + TN)$
Positive Predictive Value (PPV), Precision	What fraction of those predicted positives are actually positive?	$PPV = \frac{TP}{TP + FP}$
F-Measure (F)	A measure that combines precision (P) and recall (R) is the harmonic mean of precision and recall	$FM = 2(\frac{P * R}{P + R})$
Accuracy (ACC)	The number of Occurrences correctly classifier	$ACC = \frac{(TP + TN)}{(TP + TN + FP + FN)}$

4.7 Evaluation of Machine Learning Classifiers Based on Extracted Features

In order to evaluate the machine learning classifier for selected features different five classification algorithms are chosen including Support Vector Machine, J-48, Random Forest, Simple Logistic Regression and Naïve Bayes. This is novel approach to detect botware applications using minimal features. The purpose behind this evaluation is to select a suitable machine learning classifier to detect botware applications on the base of selected features. This experiment compares the obtained results with the existing approach which uses almost similar number of features. For this purpose, an experimental setup is followed as given in Figure 4.8. In this experiment the main three phases are involved such as data collection, features extraction and evaluation. A huge number of features are extracted in the data collection phase from Android Applications. The second phase feature extraction will help to select the most relevant features to improve the effectiveness of the proposed approach. This analysis was done to detect the unknown botware applications. The aforementioned different types of classifiers used in the evaluation phase.

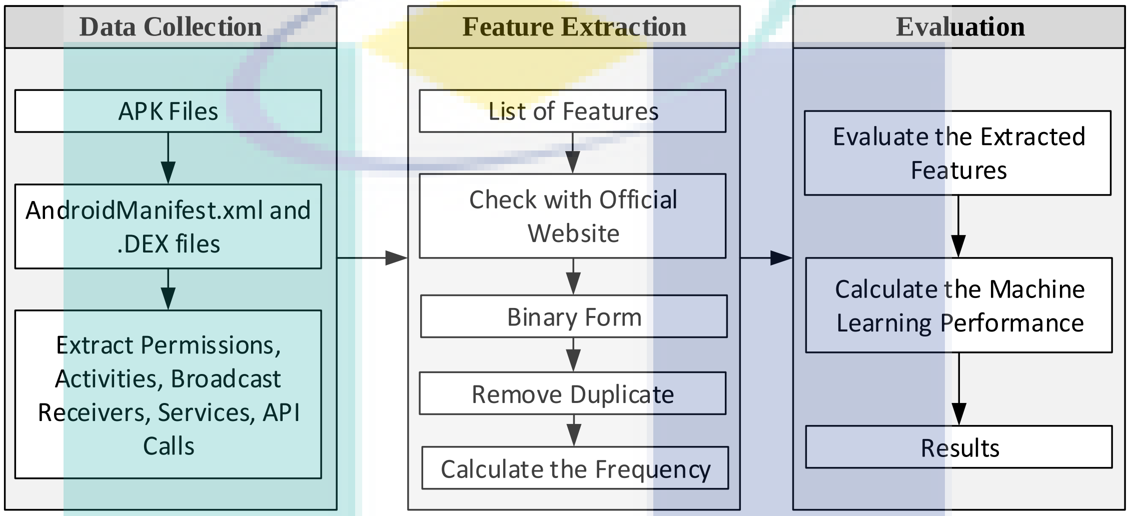

Data Collection
APK Files
AndroidManifest.xml and .DEX files
Extract Permissions, Activities, Broadcast Receivers, Services, API Calls
Feature Extraction
List of Features
Check with Official Website
Binary Form
Remove Duplicate
Calculate the Frequency
Evaluation
Evaluate the Extracted Features
Calculate the Machine Learning Performance
Results

Figure 4.8 Experimental Setup for the evaluation of Machine Learning Classifier for the selected Features

In order to collect the required features all, the input application is decompressed with the help of APK Tool. The desired features were existing in the AndroidManifest and .DEX files. The list of permissions, activities, broadcast receivers, services and API Calls are obtained from the above-mentioned files. The list of obtained features is given in Appendix B. The evaluation of the obtained results is discussed in the next sub-sections. The best results are highlighted in bold in the following tables. The comparison of the unmodified and modified approaches is given which were obtained from the above-mentioned machine learning classifiers. The results is obtained for the evaluation parameters mentioned in Section 4.6 such as true positive rate, false positive rate, precision, F-measure, and accuracy. In this experiment the data is split in 80% for training and 20% for testing samples.

4.7.1 True Positive Rate (TPR)

TPR is investigated in order to support the performance of the modified botnet detection approach based on the static analysis. Tests have been carried out for the TPR for both unmodified (without features refining) (Peiravian & Zhu, 2013) and modified (with features refining) on different five classifiers namely, Random forest, Naïve Bayes, Support Vector Machine, Simple Logistic Regression, and J-48 for all extracted features such as permissions, activities, broadcast receivers, services and API calls. The evaluation of the proposed approach is performed on the aforementioned parameters. TPR is one of these parameters that check the proportion of correctly classification of botware and benign applications by this approach. The performance of different classifiers are shown in Table 4.11 with respect to TPR for unmodified and modified approach. The higher the TPR, the better is the result. These results are generated with the aforementioned machine learning classifiers using WEKA tool.

The Minimum and Maximum values in the Table 4.11 represent the minimum and maximum ranges of generated number of TPR for different classifiers. In order to analyze the performance of proposed (modified) approach, the applications for all extracted features are executed by changing the number of k-fold cross validation for each classifier. The given TPR values for extracted features are the average of all folds of the selected classifiers. The average TPR of unmodified and modified approach are 0.87 and 0.89 respectively for permission features. The difference between unmodified and modified is 2.53% recorded, which shows the detection capability of the modified detection approach in term of TPR is better than the existing unmodified approach. Similarly, a slightly difference between unmodified and modified approaches are recorded for the remaining features as well such as 2.27% for activities, 6.25% for broadcast receivers, 4.76% for services, and 5.26% for API Calls.

Moreover, Table 4.10 describes the results of the evaluation parameters namely TPR for the activities feature. The average TPR of activities features for unmodified approach is 0.7229 and for modified approach is 0.7456. The difference between both unmodified and modified is 2.2680%, which shows the detection capability of modified detection approach in term of TPR for activities features is better than the existing unmodified approach. The average TPR of broadcast receiver features for unmodified approach is 0.6962 and for modified approach is 0.7587. However, random forest generated best results in both unmodified and modified approach. The difference between the average values of both approaches is 6.2540%, which shows the detection capability of the modified detection approach in term of TPR is better than the existing unmodified approaches. The average TPR of services features for unmodified and modified approaches are 0.6521 and 0.6997 respectively. The difference between both unmodified and modified is 4.7580 %, which shows the detection capability of our modified detection approach in term of TPR of receiver is better than the existing unmodified approach.

In the Table 4.10, the average TPR of API Calls feature for unmodified approach is 0.7008 and for the modified approach is 0.7534. However, support vector machine generated best results in modified approach while random forest generated best result for unmodified approach. The difference between both approaches is 5.2620%, which shows the detection capability of modified detection approach in term of API call’s TPR is better than the existing unmodified approach.

Figure 4.9 illustrates the whole results of the modified and unmodified approaches for the selected classifiers with respect to permissions, activities, broadcast receivers, services and API calls. The X-axis shows the TPR for each classifiers that presents on Y-axis. The TPR value approaches to one (1) is consider the best result while approaching to zero (0) will be the worst results. The bar graph shows that the Random Forest generated best results for permissions, broadcast receivers and API calls. However, SVM generated best results for the activities and services

Table 4.10 TPR comparison for unmodified and modified approach using all features for selected classifiers

	Features	**Random Forest**	**Naïve Bayes**	**SVM**	**SLR**	**J-48**	**Minimum**	**Maximum**	**Means**	**Median**	**Std. Deviation**	**Confidence Interval**
Unmodified	Permissions	**0.8954**	0.8501	0.8793	0.8570	0.8754	0.8501	0.8954	0.8714	0.8754	0.0181	0.0225
	Activities	0.7239	0.7112	0.7231	0.7023	**0.7541**	0.7023	0.7541	0.7229	0.7231	0.0196	0.0243
	Broadcast Receivers	**0.8062**	0.6412	0.7508	0.6301	0.6527	0.6301	0.8062	0.6962	0.6527	0.0781	0.0969
	Services	0.6221	**0.7020**	0.6912	0.6232	0.6220	0.6220	0.7020	0.6521	0.6232	0.0408	0.0507
	API Calls	0.7011	**0.7265**	0.6921	0.6921	0.6921	0.6921	0.7265	0.7008	0.6921	0.0149	0.0185
Modified	Permissions	**0.9114**	0.8802	0.8912	0.9000	0.9011	0.8802	0.9114	0.8968	0.9000	0.0117	0.0145
	Activities	0.7327	0.7250	**0.7918**	0.7111	0.7674	0.7111	0.7918	0.7456	0.7327	0.0331	0.0411
	Broadcast Receivers	**0.8567**	0.6600	0.8180	0.6667	0.7923	0.6600	0.8567	0.7587	0.7923	0.0901	0.1118
	Services	0.7150	0.6753	**0.7319**	0.7142	0.6620	0.6620	0.7319	0.6997	0.7142	0.0296	0.0367
	API Calls	**0.7630**	0.7550	0.7580	0.7533	0.7377	0.7377	0.7630	0.7534	0.7550	0.0095	0.0118

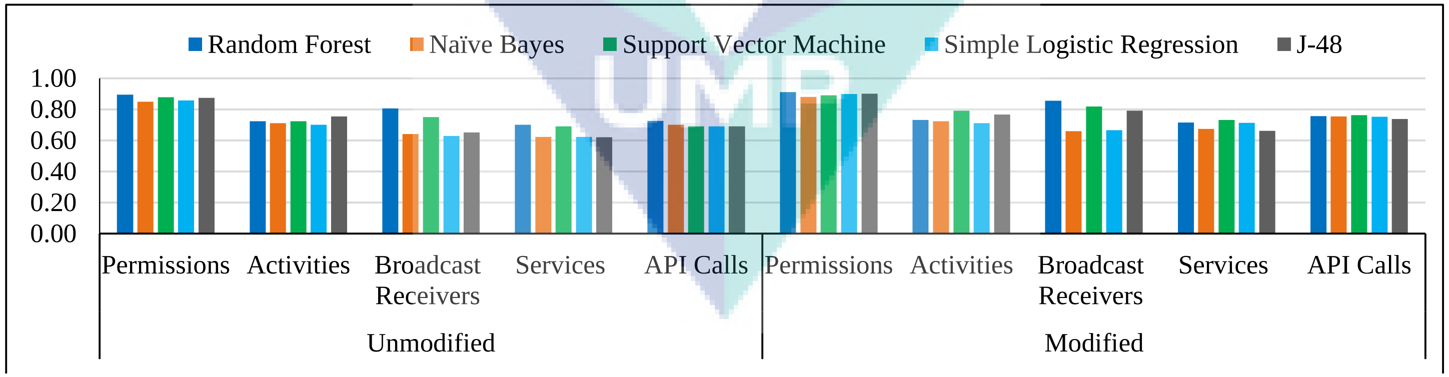


Figure 4.9 Number of TPR for unmodified and modified detection approach in term of Selected Features for all classifiers

4.7.2 False Positive Rate (FPR)

FPR is investigated in order to evaluate the detection performance of the modified botnet detection approach based on static analysis. Tests have been carried out for FPR for both unmodified (without features refining component) and modified (with features refining component) on different five classifiers namely, Random forest, Naïve Bayes, Support Vector Machine, Simple Logistic Regression, and J-48 for all extracted features such as permissions, activities, broadcast receivers, services and API calls. The evaluation of the proposed approach is performed on the aforementioned parameters. FPR is one of the parameters that check the proportion of incorrectly classification of botware and benign applications by the approach. The value of FPR that approach zero is considered best result while the value that approaches to one is considered worst performance. The lower the value of FPR, the better is the result. Table 4.11 shows the FPR of selected feature of the modified and unmodified botnet detection approach. These results are generated with the aforementioned machine learning classifiers using WEKA tool.

The Minimum and Maximum values in the Table 4.11 represent the minimum and maximum FPR ranges for selected classifiers. Using the K-fold cross validation method, the performance of proposed (modified) approach is analyzed for all extracted features. In these experiments the value of k-fold cross validation are changed for each classifier. The mentioned FPR values for extracted features are the average of all folds. The average FPR value for permission feature of unmodified and modified approach are 0.1410 and 0.0749 respectively. Hence the difference between unmodified and modified is 46.86% recorded, which shows the detection capability of the modified detection approach in term of FPR is better than the existing unmodified approach. Furthermore, there are some difference recorded between unmodified and modified approaches for the remaining features such as 35.99% for activities, 44.60% for broadcast receivers, 22.85% for services, and 22.91% for API Calls. The worst FPR is generated for services features. The FPR for services features using modified approach is 0.2345 while using the unmodified approach the FPR is 0.3040. The difference between both are 22.85%.

Table 4.11 FPR comparison for unmodified and modified approach using all features for selected classifiers

	Features	Random Forest	Naïve Bayes	SVM	SLR	J-48	Minimum	Maximum	Means	Median	Std. Deviation	Confidence Interval
Unmodified	Permissions	**0.0849**	0.0991	0.1607	0.2758	0.0847	0.0847	0.2758	0.1410	0.0991	0.0816	0.1013
	Activities	0.2187	0.3248	**0.1494**	0.2456	0.2798	0.1494	0.3248	0.2437	0.2456	0.0660	0.0819
	Broadcast Receivers	**0.0913**	0.6100	0.1293	0.6201	0.1261	0.0913	0.6201	0.3154	0.1293	0.2740	0.3402
	Services	0.1340	0.6620	**0.1321**	0.2006	0.3912	0.1321	0.6620	0.3040	0.2006	0.2263	0.2810
	API Calls	0.2132	0.2320	**0.2203**	0.2343	0.2223	0.2132	0.2343	0.2244	0.2223	0.0087	0.0108
Modified	Permissions	0.0626	0.0803	0.1118	**0.1100**	0.0100	0.0100	0.1118	0.0749	0.0803	0.0418	0.0519
	Activities	0.1673	0.1772	**0.1138**	0.1889	0.1326	0.1138	0.1889	0.1560	0.1673	0.0316	0.0392
	Broadcast Receivers	**0.0816**	0.3267	0.0882	0.2700	0.1069	0.0816	0.3267	0.1747	0.1069	0.1150	0.1428
	Services	**0.0519**	0.5820	0.1272	0.1452	0.2662	0.0519	0.5820	0.2345	0.1452	0.2089	0.2594
	API Calls	0.1840	0.1850	0.2400	0.1441	**0.1288**	0.1288	0.2400	0.1730	0.1840	0.0433	0.0537

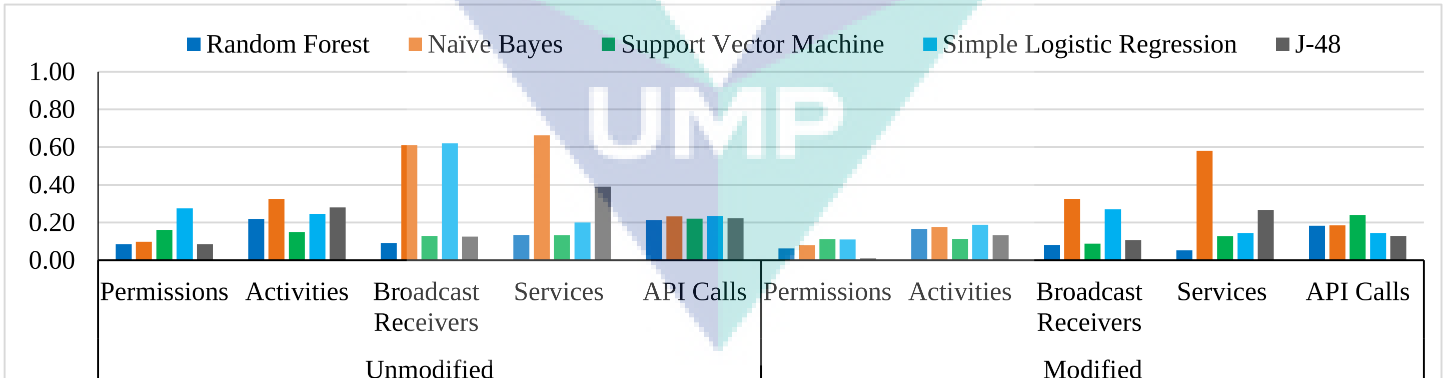


Figure 4.10 Number of FPR for unmodified and modified detection approach in term of Selected features for all classifiers

Figure 4.10 illustrates the whole results of the modified and unmodified approaches for the selected classifiers for aforementioned features. The X-axis shows the FPR for each classifier that presents on Y-axis. The FPR value approaches to one (1) is consider the worst results while approaching to zero (0) will be the best results. The bar graph shows that the Random Forest generated best results for permissions, broadcast receivers and services. However, SVM generated best results for the activities and J.48 generated good FPR for services features.

4.7.3 Precision

Precision is investigated in order to evaluate the botnet detection performance of the modified botnet detection approach based on the static analysis. Tests have been carried out for the precision for both unmodified (without features refining component) and modified (with features refining component) on different five classifiers namely, Random forest, Naïve Bayes, Support Vector Machine, Simple Logistic Regression, and J-48 for all extracted features such as permissions, activities, broadcast receivers, services and API calls. The evaluation of the proposed approach is performed on the aforementioned parameters. Precision is one of the parameters that check the proportion of incorrectly classification of botware and benign applications by the approach. The higher the precision, the better is the result. The value of precision that approaches zero is considered best result while value that approach one is considered worst performance. Table 4.12 shows the precision of permissions feature of the modified and unmodified botnet detection approach. These results are generated with the aforementioned machine learning classifiers using WEKA tool.

Table 4.12 represent the minimum, maximum, and average precision ranges for selected classifiers using the K-fold cross validation method. The mentioned precision values for extracted features are the average of all folds. The obtained average precision value of unmodified and modified approaches for permission feature is 0.8219 and 0.8850 respectively. Hence the difference between both is 7.70% recorded. It shows the detection capability of the modified detection approach in term of FPR is better than the existing unmodified approach. Similarly, there are also some difference recorded between unmodified and modified approaches for the remaining features that is 4.94% for activities, 26.94% for broadcast receivers, 21.73% for services, and 2.49% for API Calls. The worst precision is generated for API Calls features that is 0.8332. There is only 2.48% improvement occurred between both the approaches.

Figure 4.11 depicts the whole results of the modified and unmodified botnet detection approaches for the selected classifiers for aforementioned features. The X-axis shows the precision for each classifier that presents on Y-axis. The precision value approaches to one (1) is consider the best results while approaching to zero (0) will be the worst results. The bar graph shows that the Random Forest generated best results for permissions, broadcast receivers and services that is 0.9127, 0.8940 and 0.8632 respectively. However, for the activities feature Naïve Bayes generated best result that is 0.8809 and SVM generated good precision result for services features. Hence it is concluded that Random Forest is the best classifier in term of precision values for all extracted features.

Table 4.12 Precision comparison for unmodified and modified approach using all features for selected classifiers

	Features	Random Forest	Naïve Bayes	SVM	SLR	J-48	Minimum	Maximum	Means	Median	Std. Deviation	Confidence Interval
Unmodified	Permissions	**0.9119**	0.8517	0.7217	0.8000	0.8244	0.7217	0.9119	0.8219	0.8244	0.0698	0.0867
	Activities	**0.8226**	0.8079	0.7236	0.7566	0.8101	0.7236	0.8226	0.7842	0.8079	0.0423	0.0525
	Broadcast Receivers	**0.7210**	0.5430	0.7210	0.5362	0.6224	0.5362	0.7210	0.6287	0.6224	0.0908	0.1127
	Services	0.6136	0.5380	**0.7103**	0.7017	0.5380	0.5380	0.7103	0.6203	0.6136	0.0841	0.1045
	API Calls	**0.8222**	0.8123	0.8012	0.8143	0.8149	0.8012	0.8222	0.8130	0.8143	0.0076	0.0094
Modified	Permissions	**0.9127**	0.9016	0.8555	0.8443	0.9123	0.8443	0.9127	0.8853	0.9016	0.0328	0.0408
	Activities	0.8314	**0.8809**	0.7598	0.7741	0.8684	0.7598	0.8809	0.8229	0.8314	0.0545	0.0676
	Broadcast Receivers	**0.8940**	0.7960	0.7932	0.6360	0.8714	0.6360	0.8940	0.7981	0.7960	0.1011	0.1255
	Services	**0.8632**	0.5780	0.8503	0.8461	0.6380	0.5780	0.8632	0.7551	0.8461	0.1361	0.1690
	API Calls	0.8393	0.8270	**0.8410**	0.8237	0.8348	0.8237	0.8410	0.8332	0.8348	0.0076	0.0094

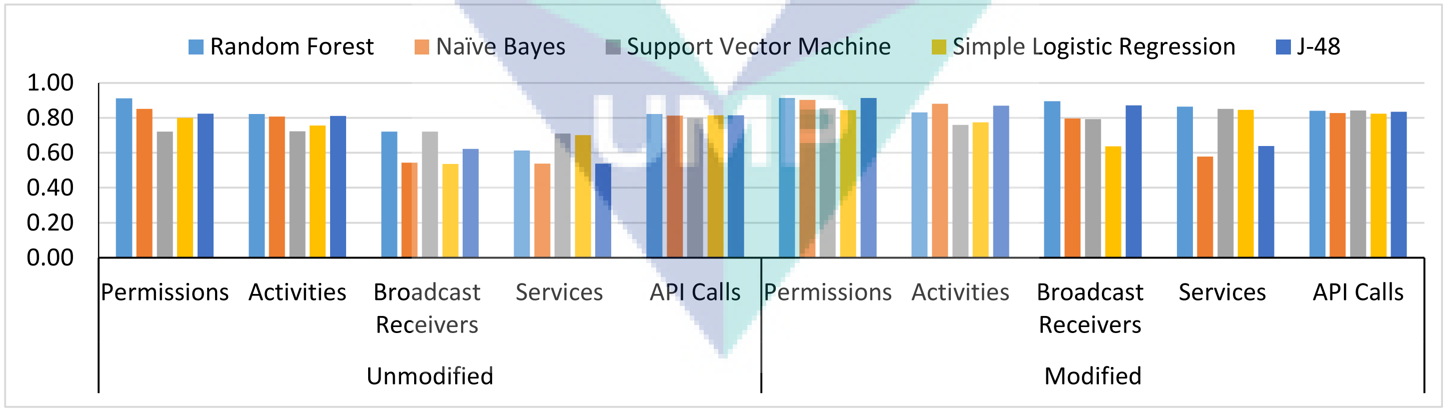


Figure 4.11 Number of Precision for unmodified and modified detection approach in term of Selected features for all classifiers

4.7.4 F-Measure

F-measure is investigated in order to evaluate the botnet detection performance of the modified botnet detection approach based on the static analysis. Tests have been carried out for the F-measure for both unmodified (without features refining component) and modified (with features refining component) on different five classifiers namely, Random forest, Naïve Bayes, Support Vector Machine, Simple Logistic Regression, and J-48 for all extracted features such as permissions, activities, broadcast receivers, services and API calls. The evaluation of the proposed approach is performed on the aforementioned parameters. F-measure is one of the parameters that check the proportion of incorrectly classification of botware and benign applications by the approach. The higher the F-measure, the better is the result. The value of F-measure that approaches zero is considered worst result while value that approaches one is considered best performance. Table 4.13 shows the F-measure of permissions feature of the modified and unmodified botnet detection approach.

The Minimum and Maximum values in the Table 4.13 represent the minimum and maximum ranges of generated results of F-Measure for different selected classifiers. In order to analyze the performance of proposed (modified) approach, the applications for all extracted features are executed by changing the number of k-fold cross validation for each classifier. The given F-Measure values for extracted features are the average of all folds of the selected classifiers. The average F-Measure of unmodified and modified approach are 0.7979 and 0.8812 respectively for permission features. The difference between unmodified and modified is 10.44% recorded. This difference shows the detection capability of the modified detection approach in term of F-Measure which is better than the existing unmodified approach.

Moreover, Table 4.13 describes the results of the evaluation parameters namely F-measure for the activities feature. The average F-Measure of activities features for unmodified approach is 0.6996 and for modified approach is 0.7580. The difference between both unmodified and modified is 8.35%. Similarly, a notable difference is seems between the both approaches for the broadcast receivers that is 18.26%. The average F-measure of services features for both approaches are 0.6335 and 0.7085 respectively. Thus, the difference between both unmodified and modified is 11.84%, which shows the detection capability of our modified detection approach in term of F-measure of receiver is better than the existing unmodified approach. Furthermore, the average F-measure of

API Calls feature for unmodified approach is 0.6727 and for the modified approach is 0.7385. The difference between both approaches is 9.78% shows the detection capability of modified detection approach in term of API call's F-measure is better than the existing unmodified approach.

Figure 4.12 illustrates the whole results of the modified and unmodified approaches for the selected classifiers with respect to permissions, activities, broadcast receivers, services and API calls. The X-axis shows the F-measure for each classifier that presents on Y-axis. The TPR value approaches to one (1) is consider the best result while approaching to zero (0) will be the worst results. The bar graph shows that the Random Forest generated best results for permissions, broadcast receivers and services. However, J-48 generated best results for the activities and API Calls in the modified detection approach.

Table 4.13 F-Measure comparison for unmodified and modified approach using all features for selected classifiers

	Features	Random Forest	Naïve Bayes	SVM	SLR	J-48	Minimum	Maximum	Means	Median	Std. Deviation	Confidence Interval
Unmodified	Permissions	0.8144	**0.8517**	0.8117	0.7000	0.8119	0.7000	0.8517	0.7979	0.8119	0.0573	0.0712
	Activities	0.7004	0.6985	0.6892	0.6763	**0.7336**	0.6763	0.7336	0.6996	0.6985	0.0213	0.0264
	Broadcast Receivers	**0.7722**	0.4371	0.7336	0.4371	0.7092	0.4371	0.7722	0.6178	0.7092	0.1665	0.2068
	Services	**0.6755**	0.6147	0.6235	0.6392	0.6147	0.6147	0.6755	0.6335	0.6235	0.0255	0.0317
	API Calls	**0.6803**	0.6656	0.6856	0.6570	0.6748	0.6570	0.6856	0.6727	0.6748	0.0115	0.0142
Modified	Permissions	**0.9358**	0.9110	0.8832	0.7633	0.9128	0.7633	0.9358	0.8812	0.9110	0.0685	0.0851
	Activities	0.7597	0.7499	0.7634	0.7259	**0.7912**	0.7259	0.7912	0.7580	0.7597	0.0236	0.0293
	Broadcast Receivers	**0.8602**	0.5250	0.8461	0.6250	0.7971	0.5250	0.8602	0.7307	0.7971	0.1483	0.1841
	Services	**0.7517**	0.7281	0.7359	0.6800	0.6470	0.6470	0.7517	0.7085	0.7281	0.0436	0.0541
	API Calls	0.7457	0.7330	**0.7510**	0.7224	0.7402	0.7224	0.7510	0.7385	0.7402	0.0112	0.0139

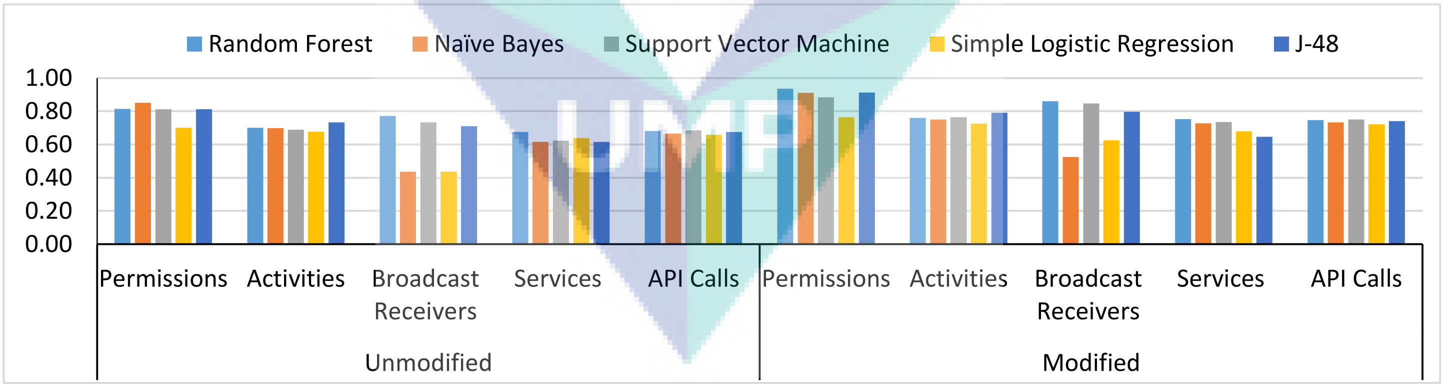


Figure 4.12 Number of F-measure for unmodified and modified detection approach in term of Selected features for all classifiers

4.7.5 Accuracy

Accuracy is investigated in order to evaluate the botnet detection performance of the modified botnet detection approach based on static analysis. Tests have been carried out for the accuracy for both unmodified (without features refining component) and modified (with features refining component) on different five classifiers namely, Random forest, Naïve Bayes, Support Vector Machine, Simple Logistic Regression, and J-48 for all extracted features such as permissions, activities, broadcast receivers, services and API calls. The evaluation of the proposed approach is performed on the aforementioned parameters. Accuracy is one of the parameters that check the proportion of incorrectly classification of botware and benign applications by the approach. The higher the accuracy, the better is the result. When the values of accuracy approach to zero, then this is considered as the best result while when the value of accuracy approaches one, it is considered as worst performance. Table 4.14 shows the correctness of accuracy feature of the modified and unmodified botnet detection approach.

The Minimum and Maximum values in the Table 4.14 represent the minimum and maximum ranges of generated accuracy for different classifiers using unmodified and modified detection approaches. In order to analyze the performance of proposed (modified) approach, the applications for all extracted features are executed by changing the number of k-fold cross validation for each classifier. The given values accuracy for all extracted features are the average of 10 folds of the selected classifiers. The average accuracy for permission feature of unmodified and modified approach are 84.74% and 89.82% respectively. The difference between unmodified and modified is 6.00% recorded, which shows the detection capability of the modified detection approach in term of TPR is better than the existing unmodified approach. Similarly, a notably difference between unmodified and modified approaches are recorded for the remaining features such as 2.07% for activities, 11.42% for broadcast receivers, 8.37% for services, and 3.30% for API Calls. Figure 4.13 shows that the Random Forest generated best results for all selected features except activities.

Table 4.14 Accuracy comparison for unmodified and modified approach using all features for selected classifiers

	Features	Random Forest	Naïve Bayes	SVM	SLR	J-48	Minimum	Maximum	Means	Median	Std. Deviation	Confidence Interval
Unmodified	Permissions	**92.44**	86.12	81.02	85.00	79.11	79.11	92.44	84.74	85.00	5.17	6.42
	Activities	80.39	80.34	71.19	65.84	**80.46**	65.84	80.46	75.58	80.03	6.72	8.35
	Broadcast Receivers	**85.00**	68.41	80.32	68.44	72.75	68.41	85.00	74.98	72.75	7.41	9.20
	Services	**71.36**	62.28	67.23	62.32	68.18	62.28	71.36	66.27	67.23	3.94	4.89
	API Calls	**76.24**	73.57	72.34	71.42	72.33	71.42	76.24	73.18	72.34	1.87	2.33
Modified	Permissions	**93.40**	91.12	87.03	86.93	90.64	86.93	93.40	89.82	90.64	2.80	3.47
	Activities	80.42	66.02	79.91	77.18	**80.71**	66.02	80.71	77.15	79.91	6.46	8.02
	Broadcast Receivers	**86.68**	81.02	81.80	83.02	85.23	81.02	86.68	83.55	83.02	2.36	2.94
	Services	**78.86**	68.18	74.48	66.18	71.42	66.18	78.86	71.82	71.42	5.05	6.27
	API Calls	**78.26**	74.57	76.29	73.54	75.33	73.54	78.26	75.60	75.33	1.80	2.23

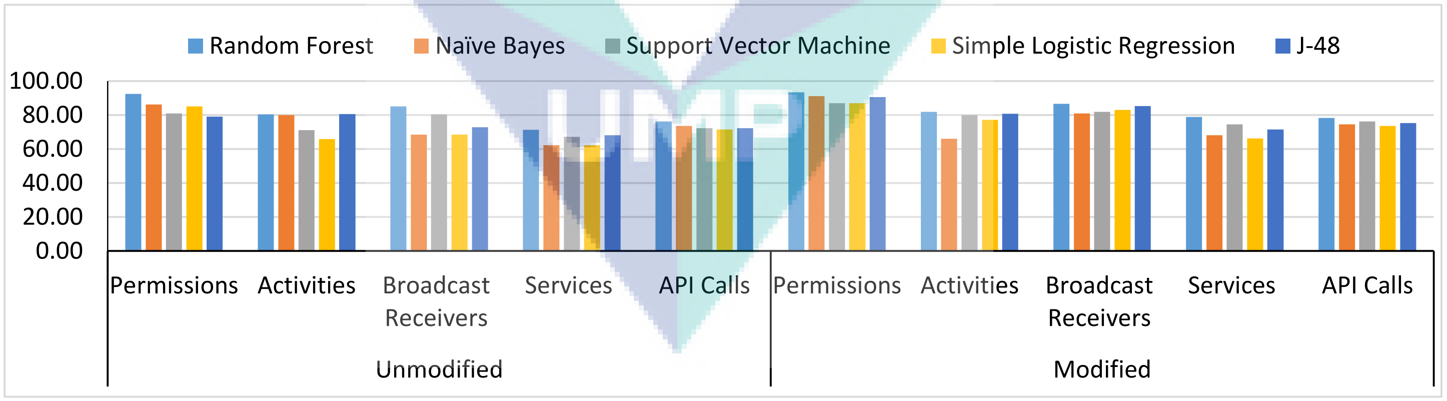


Figure 4.13 Accuracy for unmodified and modified detection approach in term of Selected features for all classifiers

Table 4.15 T and P values for Unmodified and Modified approaches in term of TPR

	Unmodified	Modified	Unmodified	Modified	Unmodified	Modified	Unmodified	Modified	Unmodified	Modified
	Permissions		**Activities**		**Broadcast Receivers**		**Services**		**API Calls**	
RF	0.8954	0.9114	0.7239	0.7827	0.8062	0.8567	0.7020	0.7150	0.7265	0.7580
NB	0.8501	0.8802	0.7112	0.7250	0.7412	0.6600	0.6221	0.6753	0.7011	0.7550
SVM	0.8793	0.8912	0.7231	0.7918	0.7508	0.8180	0.6912	0.7319	0.6921	0.7130
SLR	0.8570	0.9000	0.7023	0.7111	0.7301	0.6667	0.6232	0.7142	0.6921	0.7033
J-48	0.8754	0.9011	0.7541	0.7774	0.6527	0.7923	0.6220	0.6620	0.6921	0.7177
T Test	2.6246		1.9801		2.6186		2.1112		2.1797	
P Test	0.0152		0.0500		0.0154		0.0339		0.0304	

Table 4.16 T and P values for Unmodified and Modified approaches in term of FPR

	Unmodified	Modified	Unmodified	Modified	Unmodified	Modified	Unmodified	Modified	Unmodified	Modified
	Permissions		**Activities**		**Broadcast Receivers**		**Services**		**API Calls**	
RF	0.0849	0.0626	0.2187	0.1673	0.0913	0.0816	0.1340	0.0519	0.2132	0.1840
NB	0.0991	0.0803	0.3248	0.1772	0.6100	0.1267	0.2620	0.1820	0.2320	0.1850
SVM	0.1607	0.1118	0.1494	0.1138	0.1293	0.0882	0.1321	0.1272	0.2203	0.2400
SLR	0.2758	0.0100	0.2456	0.1889	0.6201	0.1700	0.2006	0.1452	0.2343	0.1441
J-48	0.0847	0.0100	0.2798	0.1326	0.1261	0.1069	0.2912	0.1462	0.2223	0.1288
T-Value	2.0699		2.6817		1.9843		1.9862		2.4345	
P-Value	0.0361		0.0139		0.0467		0.0480		0.0205	

Table 4.17 T and P values for Unmodified and Modified approaches in term of Precision

	Unmodified	Modified	Unmodified	Modified	Unmodified	Modified	Unmodified	Modified	Unmodified	Modified
	Permissions		Activities		Broadcast Receivers		Services		API Calls	
RF	0.9119	0.9127	0.8226	0.8814	0.7210	0.8940	0.6136	0.8632	0.8222	0.8393
NB	0.8517	0.9016	0.8079	0.8809	0.5430	0.7960	0.5380	0.5780	0.8123	0.8270
SVM	0.7217	0.8555	0.7236	0.7898	0.7210	0.7932	0.7003	0.8503	0.8012	0.8410
SLR	0.8000	0.8743	0.7566	0.7874	0.5362	0.6360	0.7017	0.8561	0.8143	0.8237
J-48	0.8244	0.9223	0.8101	0.8684	0.6224	0.8714	0.5380	0.6480	0.8149	0.8348
T-Value	2.1227		1.9926		2.7876		1.9882		4.2129	
P-Value	0.0332		0.0407		0.0118		0.0409		0.0015	

Table 4.18 T and P values for Unmodified and Modified approaches in term of F-Measure

	Unmodified	Modified	Unmodified	Modified	Unmodified	Modified	Unmodified	Modified	Unmodified	Modified
	Permissions		Activities		Broadcast Receivers		Services		API Calls	
RF	0.8144	0.9358	0.7004	0.7297	0.7722	0.8602	0.6755	0.7517	0.7003	0.7457
NB	0.8517	0.9110	0.6985	0.7299	0.6371	0.7250	0.6147	0.7281	0.7156	0.7330
SVM	0.8117	0.8832	0.6892	0.7334	0.7336	0.8461	0.6235	0.7359	0.7356	0.7510
SLR	0.7000	0.7633	0.6763	0.7159	0.4371	0.7250	0.6392	0.6800	0.7170	0.7224
J-48	0.8119	0.9128	0.7336	0.7412	0.7092	0.7971	0.6147	0.6470	0.7248	0.7402
T-Value	2.0849		2.9375		2.0119		3.3221		2.5839	
P-Value	0.0352		0.0100		0.0395		0.0052		0.0162	

Table 4.19 T and P values for Unmodified and Modified approaches in term of Accuracy

	Unmodified	Modified	Unmodified	Modified	Unmodified	Modified	Unmodified	Modified	Unmodified	Modified
	Permissions		**Activities**		**Broadcast Receivers**		**Services**		**API Calls**	
RF	92.44	93.40	77.39	81.93	85.00	86.68	71.36	78.86	76.24	78.26
NB	86.12	91.12	65.03	68.02	68.41	81.02	62.28	68.18	73.57	74.57
SVM	81.02	87.03	71.19	84.91	80.32	81.80	67.23	74.48	72.34	76.29
SLR	85.00	86.93	65.84	87.18	68.44	83.02	62.32	66.18	71.42	73.54
J-48	79.11	91.64	76.46	80.71	72.75	85.23	68.18	72.42	72.33	75.33
T-Value	1.99		2.22		2.46		2.01		2.08	
P-Value	0.04		0.02		0.02		0.04		0.03	

From the obtained results, it is clearly understood that the Android security relies on the given features. Permissions are the most prominent features which can control the applications to access the software components. Besides this it can control the applications from accessing the hardware components as well. The T-values and P-values for all the extracted features are given in Table 4.15 to Table 4.19. On the base of T values and P values it is concluded that the proposed approach is significant.

4.8 Evaluation of Machine Learning Classifiers based on Unique Patterns

In order to evaluate the machine learning classifier based on unique patterns (UP) a series of experiments are conducted using cross-fold validation technique. Different five classification algorithms are chosen to evaluate the performance of proposed approach including Support Vector Machine (SVM), J-48, Random Forest, Simple Logistic Regression (SLR) and Naïve Bayes. In order to select a suitable machine learning classifier to detect botware applications based selected unique patterns this evaluation is performed. For this purpose, an experimental setup is followed as given in Figure 4.8. This experiment compares the obtained results with the existing approach which uses almost similar number of features. There are three main phases are involved in these experiments such as data collection, features extraction and evaluation. In the data collection phase, a huge number of features are collected from Android Applications. The second phase feature extraction will help to select the most relevant features on the base of their frequencies. In this phase a unique pattern of features is generated on the bases of their usage frequencies. This analysis was done to detect the unknown botware applications. The aforementioned diverse types of classifiers used in the evaluation phase.

The APK tool is used for decompressing the input Android applications for the desired features which are existing in the Manifest and Dex files. The list of permissions, activities, broadcast receivers, services and API Calls are obtained from the above-mentioned files. The complete list of extracted features is given in Appendix B. Using the frequencies of these features a complete list of unique patterns are generated. The list of unique patterns is given in Appendix C. The comparison of the unmodified and modified approaches is given which were obtained from the above-mentioned machine learning classifiers. The results are obtained for the evaluation parameters mentioned in Section 4.6 such as true positive rate, false positive rate, precision, F-measure, and accuracy. In this experiment the data is split in 80% for training and 20% for testing samples. Figure 4.14 shows the experimental setup for the evaluation of Machine Learning Classifier for the Unique Patterns (UP).

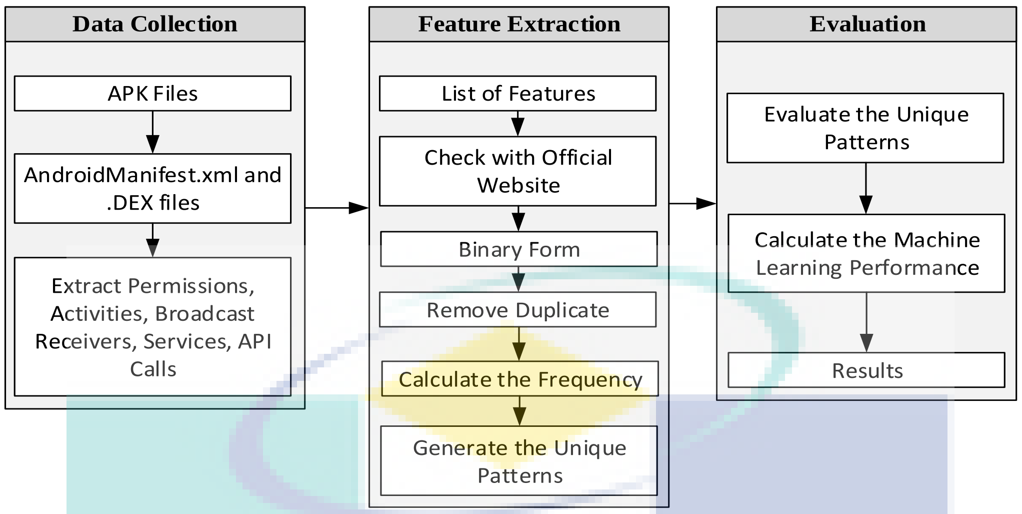

Data Collection
APK Files
AndroidManifest.xml and .DEX files
Extract Permissions, Activities, Broadcast Receivers, Services, API Calls
Feature Extraction
List of Features
Check with Official Website
Binary Form
Remove Duplicate
Calculate the Frequency
Generate the Unique Patterns
Evaluation
Evaluate the Unique Patterns
Calculate the Machine Learning Performance
Results

Figure 4.14 Experimental Setup for the evaluation of Machine Learning Classifier for the Unique Patterns (UP)

4.8.1 Results

In this experiment the evaluation of all the selected machine learning classifiers is performed by splitting the dataset in training and testing set with a ratio of 80% and 20% respectively. The detail about under examined dataset is given in Section 4.3.1. The obtained results are summarized in Table 4.21. The TPR, Precision and F-measure that approach to 1 will be consider the best results while approaching to 0 will be consider the worst results. However, in the FPR it is opposite such as the result will be considered best when the FPR approaching to 0 while it will be considered worst when it approaching to 1. Furthermore, the accuracy is calculated in percentage (%). The accuracy when it approaching to 100 will be considered the best results while approaching to 0 will be considered the worst result. By considering the above statements, it can be seen from the Table 4.20 that random forest generated best results in term of accuracy among other classifiers with an average of 97.28% accuracy. The best results are highlighted in the table.

Table 4.20 shows that the average TPR using the unique patterns for Random Forest classifier is 0.89 while 0.97 for unmodified and modified approaches respectively. The TPR for the remaining classifiers are nearly same that is 0.80 without using the feature refining component. Moreover, Naïve Bayes generated 0.87, SVM and J48 generated the same TPR that is 0.86 and SLR generated low TPR as compare to other classifier that is 0.85 by using the refining component. The overall Random Forest generated a higher TPR than other classifiers. Furthermore, the FPR values generated by all machine learning classifiers are given in the Table 4.21. By contrast, the FPR values of Random forest, Naïve Bayes, SVM, SLR and J-48 are 0.18, 0.39, 0.16, 0.32, and 0.22 respectively without using the features refining component. However, the FPR Values of Random forest, Naïve Bayes, SVM, SLR and J-48 are 0.04, 0.20, 0.14, 0.17 and 0.17 respectively by using the features refining component. In term of FPR value, Random Forest generated 0.04 value which is the lowest FPR, it shows that the Random Forest is the best classifiers to be used in this approach.

Similarly, the precision and F-measure in the 10-fold cross validation are depicted in the Table 4.20 and Figure 4.15. On the average, the precision of unmodified approach for Random Forest, Naïve Bayes, SVM, SLR and J-48 are 0.87, 0.80, 0.82, 0.78, and 0.82 respectively. The precision describes the performance of the machine learning classifiers in the proposed approach. Thus, in the obtained results for the proposed approach, the precision values of Random Forest (0.89) is more than the other classifiers. The precision values of Naïve Bayes, SVM, SLR and J-48 using feature refining component are 0.84, 0.85, 0.83, and 0.86 respectively. The F-measure values for selected classifiers without using feature refining component are Random Forest (0.94), Naïve Bayes (0.80), SVM (0.81), SLR (0.84), and J-48 (0.80). While with the using of refine component the F-measure values are slightly increased. Thus, the new values of all these selected classifiers are 0.95, 0.88, 0.89, 0.90, and 0.87 respectively.

It can be observed that random forest achieved maximum F-measure value as compare to other classifiers. Figure 4.15 shows the bar graph of TPR, FPR, Precision and F-measure for unmodified and modified approaches. The accuracy is the last parameters to evaluate the performance of proposed approach (modified approach).

Table 4.20 Results for unmodified and modified detection approach in term of Unique Patterns for all classifiers

	TPR		FPR		Precision		F-Measure		Accuracy	
	Unmodified	**Modified**	**Unmodified**	**Modified**	**Unmodified**	**Modified**	**Unmodified**	**Modified**	**Unmodified**	**Modified**
Random Forest	0.89	**0.97**	0.18	**0.04**	0.87	**0.89**	0.94	**0.95**	81.09	**97.28**
Naïve Bayes	0.81	0.87	0.39	0.20	0.80	0.84	0.80	0.88	82.08	86.18
SVM	0.80	0.86	0.16	0.14	0.82	0.85	0.81	0.89	84.42	89.90
SLR	0.80	0.85	0.32	0.17	0.78	0.83	0.84	0.90	75.60	82.37
J-48	0.81	0.86	0.22	0.17	0.82	0.86	0.80	0.87	78.57	80.67
Minimum	0.80	0.85	0.16	0.04	0.78	0.83	0.80	0.87	75.60	80.67
Means	0.82	0.88	0.25	0.14	0.82	0.85	0.84	0.90	80.35	87.28
Median	0.81	0.86	0.22	0.17	0.82	0.85	0.81	0.89	81.09	86.18
Maximum	0.89	0.97	0.39	0.20	0.87	0.89	0.94	0.95	84.42	97.28
Std. Deviation	0.04	0.05	0.10	0.06	0.03	0.02	0.06	0.03	3.39	6.63
Confidence Interval	0.03	0.04	0.09	0.05	0.03	0.02	0.05	0.03	2.97	5.81
T value	2.14		2.12		1.98		2.00		2.08	
P value	0.03		0.03		0.04		0.04		0.03	

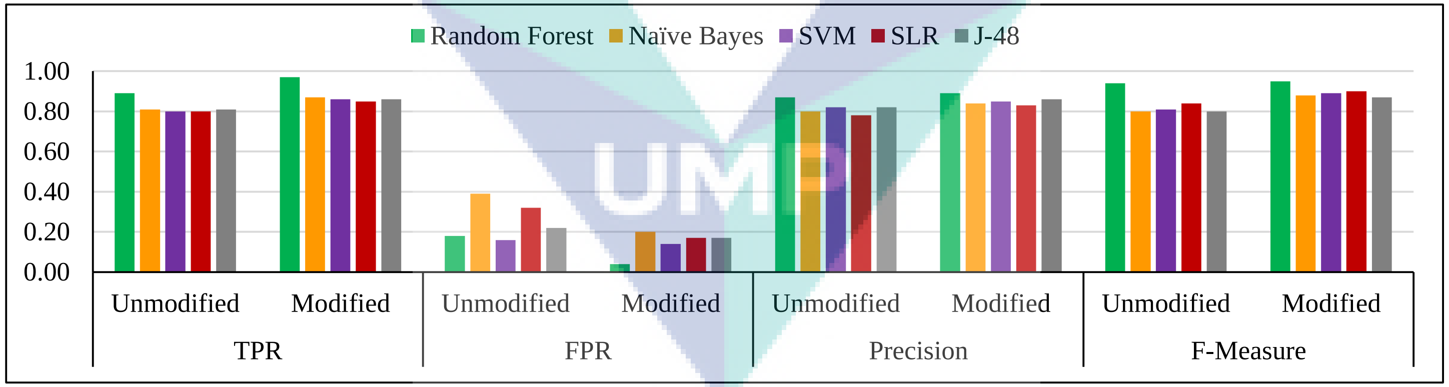
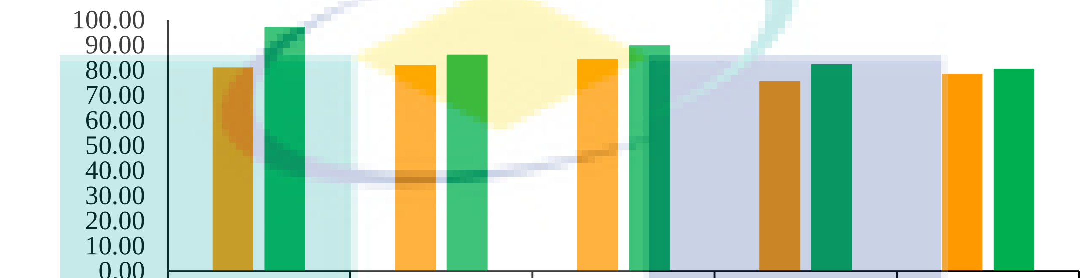


Figure 4.15 Results for unmodified and modified detection approach in term of Unique Patterns for all classifiers

Figure 4.16 depicts the results of the experiments conducted to measure the accuracy rates of the selected classifiers on both modified and unmodified approaches for unique patterns in percent. Ultimately, it determines the best possible algorithm to classify the proposed approach. The results show that Random Forest, Naïve Bayes, SVM, SLR and J-48 classify the approach with accuracy rates of 97.28%, 86.18%, 89.90%, 82.37% and 80.67% respectively in the K-fold cross validation.



	Random Forest	Naïve Bayes	SVM	SLR	J-48
🟧 Unmodified	81.09	82.08	84.42	75.60	78.57
🟩 Modified	97.28	86.18	89.90	82.37	80.67

Figure 4.16 Machine Learning Classifier Evaluation of Modified and Unmodified approaches in term of Accuracy for selected unique patterns

4.8.2 Discussion

This study has highlighted the use of machine learning classifier which can effectively detect and analyse the Android botware applications based on unique patterns. The results obtained offered a better understanding of the information derived from examining the Android permissions, activities, broadcast receivers, services and API Calls. The crucial aspect of the Android botware detection was described in detail and the methodology was also given in detail accordingly to demonstrate how the experiment processes were conducted according to phases. This was then followed by the exploration of the machine learning classifier which was used for training and testing the dataset as well as for predicting and distinguishing the Android application as botware or benign. In this regard, five machine learning classifiers were implemented. Finally, the results obtained were discussed in above sections.

The results show that the Android security relies on the permissions, activities, broadcast receivers, services and API calls. These features can control the applications from the misuse and also from the access to the software components. Furthermore, these can control the applications to access the mobile hardware. In the meanwhile, Android security has become the major security challenge, where fine grained permissions control is necessary for Android applications. However, the Google implemented the Bouncer to analyse the submitted applications on Google play store, in this study the existing permissions, activities, broadcast receivers, services and API Calls are discovered which were used for Android security. During the installation of any application on the mobile devices, the users are given an option to accept or refuse the permissions which every application request. Many of the application they force the users to accept these permissions during the installation. Consequently, most of the users who had proceeded with the installations ignored the permissions warning, might become the target of botnet attacks. The existing work focus on the permissions model of the Android malware detection, but still the privacy leak happening in the mobile devices.

In order to address this problem, an enhanced approach using a refine feature component with additional features such as activities, broadcast receivers, services and API calls is proposed in this study. This was accomplished by analysing the above-mentioned features. The experiments conducted with these features individually and with unique patterns showed that the proposed approach had achieved a high detection rate and a low false positive rate. In addition, this study had also applied supervised learning using five classifiers including Random Forest, Naïve Bayes, SVM, SLR, and J-48 on a collection of 7035 botware and benign applications. Thus, it is concluded that the proposed approach obtained high accuracy that is 97.28% by using the unique patterns of extracted features. This accuracy suggest that the approach is capable of detecting almost all the botware applications.

4.9 Performance Analysis

The mobile devices have some common issues, and without consideration of these limitations, this technique is not beneficial to mobile devices. Some of these limitations are given in Chapter 2. In summary, mobile devices have time, power, memory, processing, data access and some other limitations (Karim. *et al.*, 2015). Having identified these limitations in mobile devices, the present technique focused on the time. However, extraction of features from a big dataset can be the cause of time and memory consuming. This can also affect the processing and communication performance of the mobile devices on which it is installed. Therefore, in the initial stage the time consumption for features extraction is examined. The program is developed in Java language, which focused on the manifest.xml and DEX classes, from which this technique extracts the given features on the priority basis. These tests are performed on the Android virtual devices and some real mobile devices which are used in the experimental.

Table 4.21 summarizes the time taken for botware and benign dataset extractions. Permission features extraction process and generating CSV files for benign training set requires 0.89 seconds. Similarly, for activities, broadcast receivers, services, and API_Calls features extraction process and generating CSV files from benign training dataset take 1.50, 1.20, 1.55 and 1.22 seconds respectively. CSV files are generated for further processing. Comparingly, the same Java code takes 305 seconds for the extraction features and generating CSV files for botware datasets.

Table 4.21 Time Comparison of Botware and Benign Datasets

Dataset	Group	Dataset Scanned and Store as comma separated values	Time (seconds)
Training dataset	Botware	Permissions Extracted and Scanned from Training Dataset and Store in Botnet_Permissions.csv	0.89
		Activities Extracted and Scanned from Training Dataset and Store in Botnet_Activities.csv	1.50
		Broadcast receivers Extracted and Scanned from Training Dataset and Store in Botnet_broadcast_receivers.csv	1.20
		Services Extracted and Scanned from Training Dataset and Store in Botnet_services.csv	1.55
		API_Calls Extracted and Scanned from Training Dataset and Store in Botnet_API_Calls.csv	1.22
	Benign	Permissions Extracted and Scanned from Training Dataset and Store in Benign_Permissions.csv	1.00
		Activities Extracted and Scanned from Training Dataset and Store in Benign_Activities.csv	1.10
		Broadcast receivers Extracted and Scanned from Training Dataset and Store in Benign_broadcast_receivers.csv	1.05
		Services Extracted and Scanned from Training Dataset and Store in Benign_services.csv	1.12
		API_Calls Extracted and Scanned from Training Dataset and Store in Benign_API_Calls.csv	1.10

Table 4.21 Continued

Dataset	Group	Dataset Scanned and Store as comma separated values	Time (seconds)
Testing Dataset	Botware	Permissions Extracted and Scanned from Training Dataset and Store in Botnet_Permissions.csv	20.89
		Activities Extracted and Scanned from Training Dataset and Store in Botnet_Activities.csv	25.12
		Broadcast receivers Extracted and Scanned from Training Dataset and Store in Botnet_broadcast_receivers.csv	20.33
		Services Extracted and Scanned from Training Dataset and Store in Botnet_services.csv	15.12
		API_Calls Extracted and Scanned from Training Dataset and Store in Botnet_API_Calls.csv	10.56

4.9.1 Comparative Analysis

The comparison of proposed approach with other related works are presented in this section. The results in this experiment demonstrate that the performance of proposed approach is comparatively more efficient and better than other Android botnet detection techniques. The results in Table 4.22 show the comparison of proposed approach with other researchers work. In this comparative study, proposed is compared with other related works by different metrics, namely approach, selected number of features, sample size, accuracy, TPR, FPR, Precision, and F-Measure. Even though, accuracy metrics have been chosen to weigh the performance in comparative work, its use could be doubtful. Similarly, using accuracy for the imbalanced sample sizes, can affect the classification performance.

Table 4.22 Performance Comparison with Prior Similar Work with respect to Accuracy

Ref	Approach	No. of Features	Botware Sample Size	Benign Sample Size	TPR	FPR	Precision	F-Measure	Accuracy (%)
Proposed approach	**Static**	**40**	**3500**	**3535**	**0.97**	**0.04**	**0.89**	**0.95**	**97.28**
(Sanz *et al.*, 2013)	Static	55	249	357	0.94	0.05	0.84	0.92	94.83
(Peiravian & Zhu, 2013)	Static	1456	1200	1200	0.92	0.12	0.81	0.79	88.25
(Yerima *et al.*, 2014a)	Static	25	1000	1000	0.90	0.10	0.82	0.80	92.10
(Rashidi & Fung, 2016)	Static	30	1200	1200	0.89	0.11	0.78	0.94	95.30

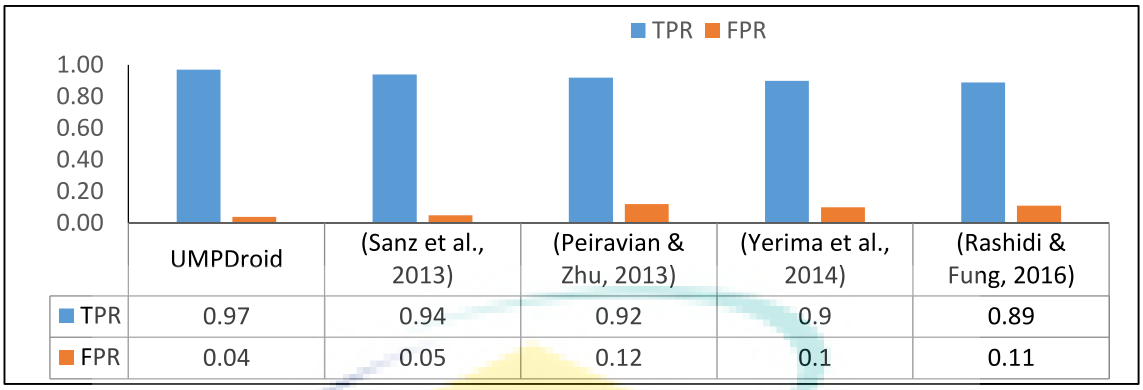
Table 4.22 Continued

Ref	Approach	No. of Features	Botware Sample Size	Benign Sample Size	TPR	FPR	Precision	F-Measure	Accuracy (%)
Minimum	---	---	---	---	0.89	0.04	0.78	0.79	88.25
Means	---	---	---	---	0.92	0.09	0.82	0.87	93.55
Median	---	---	---	---	0.92	0.10	0.82	0.92	94.83
Maximum	---	---	---	---	0.97	0.12	0.87	0.95	97.28
Std. Deviation	---	---	---	---	0.03	0.02	0.03	0.07	3.49
Confidence Interval	---	---			0.02	0.02	0.03	0.06	3.06

Source: Peiravian & Zhu, (2013); Rashidi & Fung, (2016); Sanz et al., (2013); Yerima et al., (2014a)

It may be necessary to emphasize that the proposed approach is compared with previous research studies that employed the static approach to botnet detection for the sake of fair comparison. From the analysis of the number of features, botware and benign sample sizes vis-à-vis the accuracy percentage, it is obvious that the proposed approach is the approach of choice in all counts. Besides, the proposed approach investigated a number of 40 unique patterns. However, Sanz *et al.*, (2013) used 55 features at all, Peiravian & Zhu (2013) selected 1465 number of features. Similarly, Yerima *et al.*, (2014) selected 25 numbers of features, and Rashidi & Fung, (2016) selected 30 features. In all, the number of selected unique pattern for proposed approach is also low as compare to other approaches that have an impact on proposed approach efficiency.

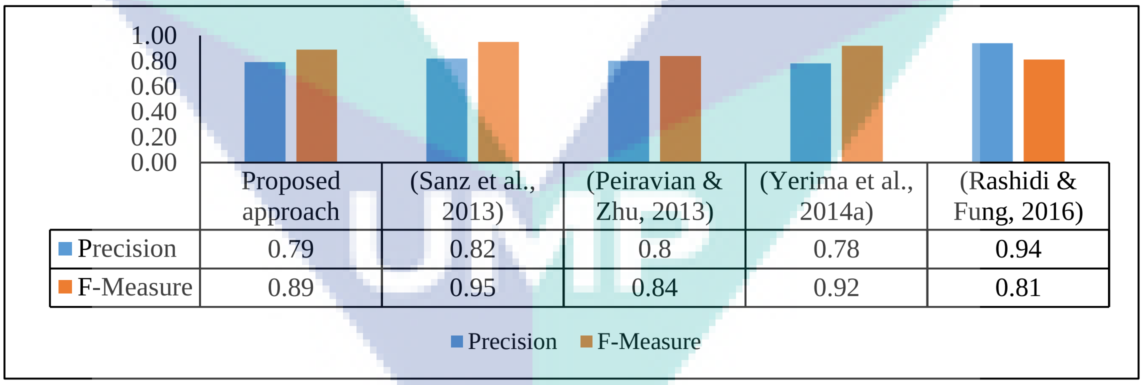
Figure 4.17 shows the comparison of TPR and FPR of proposed approach with other related works. In evaluating the TPR of the other studies, it is evident that the proposed approach obtained the best accuracy ratio of 0.97, followed by Sanz et al., (2013) with 0.94, Zhu, (2013) with 0.92, Yerima *et al*.,(2014) with 0.90 and Rashidi & Fung, (2016) with 0.89 respectively. The output of proposed approach is significant when weighed against the fact that, it had the largest sample size of 7035 which is more than three times of the sample size of the other studies. In terms of the FPR, Sanz *et al.* (2013) produced a best FPR that is 0.05, closely followed by the proposed approach with 0.09, Yerima *et al*., (2014) with 0.10 and Peiravian & Zhu, (2013) with 0.12. The performance of proposed approach is commendable when one considers its large sample size.



	UMPDroid	(Sanz et al., 2013)	(Peiravian & Zhu, 2013)	(Yerima et al., 2014)	(Rashidi & Fung, 2016)
🟦 TPR	0.97	0.94	0.92	0.9	0.89
🟧 FPR	0.04	0.05	0.12	0.1	0.11

Figure 4.17 TPR and FPR Comparison with other related work

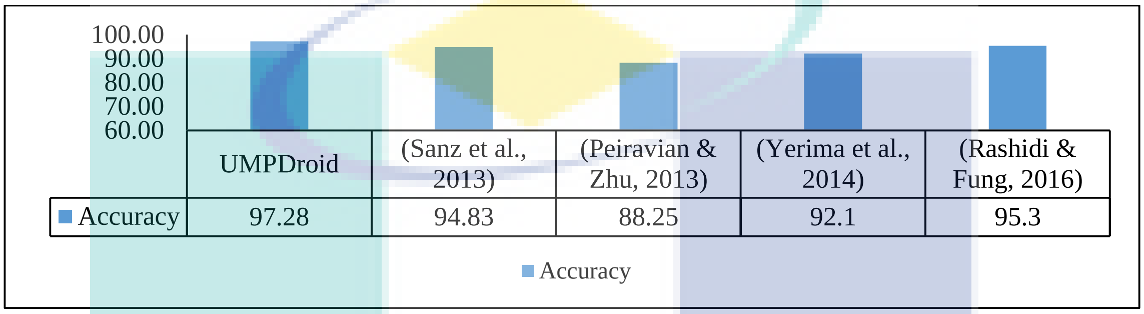
Moreover, proposed approach had the best score in Precision among the different studies with the value of 0.89 as shown in Figure 4.18. The next to proposed approach was Sanz et al., (2013) with 0.84 and Yerima et al., (2014) with 0.82. The performance value of Rashidi and Fung, (2016) was 0.78 and with a much margin from the other studies. Furthermore, F-Measure count, Rashidi & Fung, (2016) had the best result with 0.95. proposed approach did quite well, obtaining F-Measure count of 0.93. The next good result was obtained by Yerima et al., (2014) which obtained 0.80. Peiravian & Zhu, (2013) did fairly well with a result of 0.79. It is worthy to note that Sanz *et al.*, (2013), did not investigate the F-Measure in their study.



	Proposed approach	(Sanz et al., 2013)	(Peiravian & Zhu, 2013)	(Yerima et al., 2014a)	(Rashidi & Fung, 2016)
🟦 Precision	0.79	0.82	0.8	0.78	0.94
🟧 F-Measure	0.89	0.95	0.84	0.92	0.81

Figure 4.18 Precision and F-measure comparison of proposed approach approach with other related work.

Similarly, proposed approach performed very well in term of the accuracy count where a value of 97.28% was obtained compared to Rashidi & Fung, (2016) that obtained 95.30% and Sanz *et al.*, (2013), 94.83% accuracy as shown in Figure 4.19. The two lowest performance values are Yerima *et al.*, (2014) with 92.1% and Peiravian & Zhu, (2013) that had acurracy score of 88.25%. In overall, the achievements of proposed approach is satisfactory when compared with Rashidi & Fung, (2016), Sanz *et al.*, (2013), Yerima *et al.*, (2014) and Peiravian & Zhu, (2013) regarding accuracy, TPR, FPR, precision and F-Measure. The main reason for such best results of proposed approach is as a result of adding refining component and the technique of features selection. With the using of refining component proposed approach ignored those features which are not susceptible for Android botnets attacks.



	UMPDroid	(Sanz et al., 2013)	(Peiravian & Zhu, 2013)	(Yerima et al., 2014)	(Rashidi & Fung, 2016)
■ Accuracy	97.28	94.83	88.25	92.1	95.3

Figure 4.19 Accuracy Comparison of proposed approach with other related work.

4.9.2 Efficiency

In order to measure the efficiency of proposed approach, some of the random sampling are applied to the selected datasets. For random sampling, 80% is assigned to the training dataset and 20% to the test dataset. Although, similar results are obtained during the selection between 10-fold cross validation and random sampling, the former generated slightly better results than the latter. The results in Table 5.40 confirm the suitability of the Random Forest classifier for effective botware application detection within the specified feature domain. Ultimately, this is the final choice for classifier to establish in production environments.

In addition, this classifier model is implemented on the user devices to predict the scale of botnet behavior in running Android applications. Implementing this model in mobile applications would enable users to predict the correct class of an application by observing the behavior of the application. Table 4.23 shows the comparison of the efficiency between 10-fold cross validation and random sampling.

Table 4.23 Approach Efficiency Comparison between k-fold cross validation and Random Sampling

	Random Forest				
	TPR	**FPR**	**precision**	**F-Measure**	**Accuracy**
k-fold cross validation	0.98	0.05	0.93	0.89	97.28
Random Sampling	0.96	0.06	0.95	0.92	96.22
Difference (%)	2.06	18.18	2.12	3.31	1.09
Std. Deviation	0.01	0.00	0.01	0.02	0.75
Confidence Interval	0.12	0.06	0.12	0.19	6.73

Table 4.24 shows a comparison of the learning time between 10-fold cross validation and random sampling. The training times in the 10-fold cross validation range from 1.99s to 3.56s, whereas those in random sampling range from 0.03s to 8.07s. Additionally, the testing time required by the 10-fold cross validation range from 0.03s to 0.07s which are better than that of the existing machine learning based mobile malware detection solution Mobile-Sandbox. The time required to test each classifier model in the random sampling ranges from 0.02s to 3.90s. Table 4.24 shows the size of each classifier model. Size was considered to assess the feasibility of deploying the classifier model to mobile devices. The model sizes in both 10-fold cross validation and random sampling scenarios are the same. However, the model with the largest size (1.36MB) is the Random Forest model. Compared with the sizes of the Naïve Bayes (0.008 MB) and Simple Logistic Regression (1.6 MB) models in (Yerima *et al.*, 2014a), the corresponding sizes of our approach are reasonable enough, such that the approach can reside in mobile devices.

Table 4.24 Time and Size Comparisons among the selected Classifiers Models

	10-Fold Cross Validation				**Random Sampling**			
	Training Time (Seconds)	**Testing Time (seconds)**	**Model Building Time (Seconds)**	**Size (KB)**	**Training time (Second)**	**Testing Time (seconds)**	**Model Building Time (Seconds)**	**Size (KB)**
RF	3.06	0.03	1.56	1359	0.99	0.06	1.61	1359
J48	2.32	0.06	0.09	23	0.03	0.02	0.18	23
NB	3.56	0.06	0.02	8	0.94	0.09	0.02	8
SVM	1.99	0.05	8.87	479	6.97	3.90	8.85	479
SLR	3.25	0.05	5.40	1598	8.07	0.02	5.36	1598

4.9.3 Scalability

Until now, majority of the proposed solution can work either as on-device or off-device analysis systems, thereby resulting in scalability issues. However, the scalability of the proposed approach is viewed from different perspectives. There are when this solution is deployed to large-scale market stores as an offline analysis option and when the classifier is embedded into a user device for the runtime analysis of installed applications. At this time, proposed approach is deployed as an offline analysis approach to large scale market places (Google Play store) without many efforts. The required time is calculated for an Android botnet classifier model. The total time required to generate report links for the Drebin dataset (7035 Android malware) is ~18 hours, which include the uploading time, loading time to sandbox, execution time, report generation time, and network communication overhead from cloud to the host machine.

Furthermore, it is imperative to highlight that although, it is an ideal practice to process an application in sandbox, yet this is not feasible in all cases. Many factors contribute to the extension of processing time (i.e. system's peak hour, temporary disruption of service, or network communication outage). In this study, the feature extraction time is approximately an average of 10 $\pm$ minutes for execution and assigning the value against the feature vector. However, the machine learning-based classifier only consume a few seconds during the testing phase to predict the class of an application. The deployment of proposed approach logic directly into smartphone devices requires design and development of an Android application to support our machine learning classifier; such an application will be part of our future work. It is concluded that proposed approach that applies static observation is feasible for hundreds or even thousands of applications.

4.10 Summary

In this Chapter, results and evaluation of proposed approach is described for the given features and unique patterns with respect to all selected classifiers. As shown in different sections of the result, the detailed description of the results for all features and unique patterns are illustrated using appropriate Tables and Figures. However, in the evaluation section, the proposed approach is evaluated on the basis of selected features set and unique patterns. The performance evaluation of the proposed approach is carried out using 10-fold cross validation and random sampling methods. Although the proposed approach is remarkably effective in all intensity levels, the findings are more significant when the number of applications is high. The evaluation results of 97.28% accuracy are achieved. The supportive results of real time experiment and statistical modelling unveil nature of the approach, its usability and successful adoption in real scenarios.

CHAPTER 5

CONCLUSION

5.1 Overview

This chapter presents the overall conclusion of this study with the emphasis on the qualitative features of the proposed approach (Android botnet detection approach). The concluding chapter reported on the aim and objectives set out for the research in the first chapter of this thesis. As a next step, this study identified the future research work and highlighted the research contributions. The rest of this chapter is organized as follows. In Section 5.2 the research objectives of this study listed in Chapter 1 are re-examined while Section 5.3 illustrates the contribution of this research study. Section 5.4 elaborates the open issues and future work of this study and the future research directions are highlighted for further enhancement.

5.2 Review of Research Objectives

The primary aim of this study is to detect botnet attacks in Android based smartphones; while maintaining the performance to solve the problem of existing static detection techniques by adding an additional component. The research aim was broken down into three distinct objectives in Section 1.5.

The first objective was to investigate and critically analyze the current state-of-the-art botnet attacks and their detection techniques such that insight is gained leading to their detection and performance limitations. This research objective was accomplished by a thorough review of the most credible work published in articles. These were harvested from online scholarly digital libraries, such as IEEE, ACM, Elsevier, and Web of Science via the Universiti Malaysia Pahang access portal. Thorough browsing of the recent literature in the journals and conferences on Android botnet attacks in smartphones, techniques for conducting Android botnet attacks in smartphones and detection techniques for these attacks are searched and reviewed accordingly. This helped with the organization of the recent work, devised proposed taxonomy, and provided a qualitative comparison for Android botnet attacks and detection techniques for these attacks, hence, this objective is achieved in the Chapter 2 of this research.

The main purpose of this thorough study is on investigation of botnet detection techniques with the respect to perspective of dynamic, hybrid and static analysis approaches. Findings of the investigation revealed that dynamic and hybrid approaches are costly as they need more amount of battery and computational power. Furthermore, it is discovered that the existing static detection techniques based on limited features has problem in terms of detection accuracy and consequently generated low true positive and high false positive rate. Therefore, a static analysis approach with additional features was employed to improve the accuracy of Android botnet detection in Android smartphones.

The second objective of this research study was on design of a botnet detection approach which is based on static analysis with additional component and features for the detection of botnet attacks in Android smartphones. The static analysis of detection approach required less amount of battery and computational power. Moreover, the method has other benefits as a result of additional component and features, this includes ability to improve the botnet detection accuracy in smartphones, hence, and low FPR was generated. The additional features to the proposed approach detection approach which is based on static analysis approach include activities, broadcast receivers, and services; these are all extracted in the data extractor component. Specifically, additional component to the approach is feature refining that uses IG algorithm to differentiate between the botnet and benign applications based on the additional features. Design of a botnet detection approach as the second objective was accomplished in the Chapter 3.

Evaluation of this approach which is the third objective of this study was attained by assessing approach via experiment and by dividing datasets in training and testing samples in Chapter 4. This was followed with the performance of experiments for all parameters and consequently observing the results for k-fold cross validations. The performance results unveil improvement in the detection accuracy to 2.05629 %. Moreover, it improves the TPR, precision, and F-measure values by 3.14136%, 3.50877% and 1.0582% respectively while the FPR was decreased by 22.22%.

Findings of the experimental analysis were also compared with the statistical modeling to validate our proposed botnet detection approach. Hence, validation results confirm that leveraging our proposed Android botnet detection approach is able to detect botnet attacks in smartphones without affecting the performance of the smartphone and improve the detection accuracy as reported in Chapter 5.

5.3 Contribution

This section highlighted the contribution of this research study. This was reported in terms of the scholarly articles in list of publications and presented papers in the Appendix A at the end of thesis. This research produced several contributions to the body of knowledge in the following aspects.

- **Taxonomy of Android botnet Attacks:** In particular, taxonomies were characterized from the existing literature for the Android botnet attacks and detection techniques. This was achieved through the comprehensive review and critical analyzing of the PC and Smartphone botnet detection techniques from the selected state-of-the-art research work. Our comprehensive studied literature is presented in Chapter 2 and published in (Anwar *et al.*, 2017) which led to the identification of our research problem.
- **Android Botnet Detection Approach based on Static Analysis:** An approach for Android botnet detection based on static analysis is proposed. This can effectively detect botnet C&C communication features in smartphones. The proposed approach is based on the static features of Android applications. In proposed approach, features extraction is conducted by observing the static behavior of known Android botnet applications. Static features include permissions, activities, broadcast receivers, services, and API calls from Android applications. Features extraction is performed through reverse engineering process by using Androguard tool. The additional component and features are added to the existing static analysis detection approach.
- **A novel Features Refining approach:** This study proposed a novel approach for features refining in order to eliminate the repeated and irrelevant features by using a proper frequency analysis. The malicious activities of botnet applications are identified, and significant unique patterns are chosen on the base of their support value and information gain. In order to effectively identify the botware and benign applications, the proposed approach used these unique patterns.

- **Performance Evaluation and Validation:** The results of the analytical evaluation of the proposed approach are generated through Drebin dataset. Currently, Drebin is the largest available dataset for Android malware. Performance evaluation is performed on the unmodified (without feature refining) and modified (feature refining component) approaches. This was followed with development of statistical model for the evaluation parameters of proposed approach approach and for the detection of Android botnet attacks. In this research, K-fold cross validation approach is used to validate the performance of the proposed Android botnet detection approach. The results of performance evaluation and validation are presented in Chapters 4. Through the results of Statistical and schematic analysis, it was unveiled that efficiency, scalability, reliability nature of our proposed detection approach had advocated that the objectives and aim of this research study are realized and fulfilled.
- The list of related published journal and conference articles to this study are listed in Appendix A.

5.4 Open Issues and Future Work

In this research, open issues as touching the increasing security of Android devices against botnets were identified. Following the increasing number of users and emergence of cloud-computing and mobile cloud-computing platforms, the ensuing issues with respect to Android botnets are of concern to the researchers from both academia and industry alike.

Initially, cross-functional collaboration should be active among the stakeholders and researchers (government, enterprises, networks, and Internet service providers) for the Android botnet identification and confiscation tools. There is a need for clear and transparent policy on Android equipment. In addition, usage of these Android devices must be documented and standardized across the stakeholders. Moreover, the smartphone users should be aware of the way by which an Android botnet threats could be eliminated.

Next, mobile device security and risk concerns individuals from both industry and academic world. These stakeholders cannot ignore the sharp increasing demand for android devices in their respective enterprises. Not only is the demand driven by the massive adoption of consumer devices, but businesses also leverage the power of mobile computing to strengthen their value to their clients and customers, making them more agile, relevant, and able to respond to the needs of their customers.

Scanning and blocking of malicious codes in the cloud can be implemented to preempt the code, or information sharing centers can cooperate with AV vendors to identify and manage threats. When a malicious code is preempted, providers may not be able to predict how devices with diverse operating platforms receiving the code would behave with traffic. However, detection and block management of threats can be applied to blocking solutions.

Smartphones-based operating system have lower capacities as compared to PC-based operating systems, in terms of processing power, battery power, storage, and memory, which finally bound the implementation of optimum security policies.

User awareness of security threats is essential to the solution of the problem. Therefore, a specific and dedicated education and awareness campaign on pertinent risk, policies, and procedures catering to smartphone users should be introduced.

The network of infrastructure at the user level is expanding from smartphones to smart TVs, home appliances, and wearable computing equipment; thus, cooperative security mechanisms should be put in place. Apart from cloud-based or on-device analysis and monitoring systems, interconnected devices could improve their security mechanism by mutual cooperation.

Locating information leakage has become a challenging task for both on-platform and in-the-cloud analysis systems. Appropriate legislative mechanisms and several technical aspects should be explored. For example, Google App Engine, which employs a cloud-based application monitoring mechanism

In our future work, there is a plan to extend the work presented in this thesis in order to devise a hybrid on-device analysis system for the detection of botnet behavior through the use of machine-learning algorithms. For this purpose, there is a projection for the design and implementation of our sandbox with rich user interface capabilities to allow for deep code coverage and ultimately avoid all the deficiencies inherited from traditional cloud-based sandboxes.

REFERENCES

Alzahrani, A. J., Stakhanova, N., Ali, H. G., & Ghorbani, A. (2014). "Characterizing Evaluation Practicesof Intrusion Detection Methodsfor Smartphones." Journal of Cyber Security and Mobility, 3(2): 89-132.

Agrawal, R., Imielinski, T., & Swami, A. (1993). "Database Mining: A Performance Perspective." IEEE Transactions on Knowledge and Data Engineering, 5(6): 914-925.

Agrawal, R., Imieliński, T., & Swami, A. (1993). "Mining Association Rules Between Sets of Items in Large Databases." In Sigmod Record 22(2): 207-216, ACM.

Agarwal, R., & Srikant, R. (1994). "Fast Algorithms for Mining Association Rules." In 20th Very Large Data Base Conference, 487-499.

Ahmed, F., Hameed, H., Shafiq, M. Z., & Farooq, M. (2009). "Using Spatio-Temporal Information in API Calls with Machine Learning Algorithms for Malware Detection." In 2nd Workshop on Security and Artificial Intelligence, 55-62, ACM.

Al-rimy, B. A. S., Maarof, M. A., & Shaid, S. Z. M. (2018). "Ransomware Threat Success Factors, Taxonomy, and Countermeasures: A Survey and Research Directions." Computers & Security, 74: 144-166.

Alam, S., Qu, Z., Riley, R., Chen, Y., & Rastogi, V. (2017). "DROIDNATIVE: Automating and Optimizing Detection of Android Native Code Malware Variants." Computers & Security, 65: 230-246.

Alauthman, M. (2016). "An Efficient Approach to Online Bot Detection Based on a Reinforcement Learning Technique." (Doctoral Dissertation, Northumbria University).

Alazab, M., Moonsamy, V., Batten, L., Lantz, P., & Tian, R. (2012). "Analysis of Malicious and Benign Android Applications." In 32nd International Conference on Distributed Computing Systems Workshops, 608-616, IEEE.

Amazon. (2016). Amazon Applications Store "https://www.amazon.com/mobile-apps/b?ie=UTF8&node=2350149011" Accessed in July 2016.

Amos, B., Turner, H., & White, J. (2013). "Applying Machine Learning Classifiers to Dynamic Android Malware Detection at Scale." In 9th International Wireless Communications and Mobile Computing Conference, 1666-1671, IEEE.

Anagnostopoulos, M., Kambourakis, G., & Gritzalis, S. (2016). "New Facets of Mobile Botnet: Architecture and Evaluation." International Journal of Information Security, 15(5): 455-473.

Analytics, S. (2014). "Android Captures Record 85 Percent Share of Global Smartphone Shipments in Q2, 2013". http://blogs.strategyanalytics. com/WSS/post/2013/10/31/ Android-Captures-Record-81-Percent-Share-of-Global-Smartphone-Shipments-in-Q3 2013. Accessed in January 2017.

Andronio, N., Zanero, S., & Maggi, F. (2015, November). “Heldroid: Dissecting and Detecting Mobile Ransomware.” In International Workshop on Recent Advances in Intrusion Detection, 382-404, Springer.

Anwar, S., Zolkipli, M. F., Inayat, Z., Odili, J., Ali, M., & Zain, J. M. (2018). “Android Botnets: A Serious Threat to Android Devices.” Pertanika Journal of Science & Technology, 26(1): 1-37.

Apple-Inc. (2017). List of iOS Devices https://en.wikipedia.org/wiki/List_of_iOS_devices Accessed in July 2017.

AppStore, A. (2017). Apple App Store, https://support.apple.com/ Accessed in June 2017.

Apvrille, A. (2012). “Symbian Worm Yxes: Towards Mobile Botnets?” Journal in Computer Virology, 8(4): 117-131.

Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., & Siemens, C. E. R. T. (2014). “DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket.” In Network and Distributed System Security, (14), 23-26.

Aswini, A. M., & Vinod, P. (2014). “Droid Permission Miner: Mining Prominent Permissions for Android Malware Analysis.” In 5th International Conference on the Applications of Digital Information and Web Technologies 81-86, IEEE.

Babu Rajesh, V., Reddy, P., Himanshu, P., & Patil, M. U. (2015). “DROIDSWAN: Detecting Malicious Android Applications Based on Static Feature Analysis.” Computer Science and Information Technology, 163-178.

Bailey, M., Cooke, E., Jahanian, F., Xu, Y., & Karir, M. (2009). “A Survey of Botnet Technology and Defenses.” In Cybersecurity Applications & Technology Conference for Homeland Security, 299-304, IEEE.

Barrera, D., & Van Oorschot, P. (2011). “Secure Software Installation on Smartphones”. IEEE Security & Privacy, 9(3), 42-48.

Becher, M., Freiling, F. C., Hoffmann, J., Holz, T., Uellenbeck, S., & Wolf, C. (2011). “Mobile Security Catching up? Revealing the Nuts and Bolts of the Security of Mobile Devices”. In Symposium on Security and Privacy, 96-111, IEEE.

Binsalleeh, H., Ormerod, T., Boukhtouta, A., Sinha, P., Youssef, A., Debbabi, M., & Wang, L. (2010). “On the Analysis of the Zeus Botnet Crimeware Toolkit.” In 8th International Conference on Privacy, Security and Trust, 31-38, IEEE.

Bouckaert, R. R., & Frank, E. (2004). “Evaluating the Replicability of Significance Tests for Comparing Learning Algorithms.” In Pacific-Asia Conference on Knowledge Discovery and Data Mining, 3-12, Springer.

Breiman, L. (1996). “Bagging Predictors.” Machine learning, 24(2), 123-140.

Canfora, G., Mercaldo, F., & Visaggio, C. A. (2016). “An Hmm and Structural Entropy Based Detector for Android Malware: An Empirical Study.” Computers & Security, 61, 1-18.

Craig, A. N., & Shackelford, S. J. (2013). “Hacking the Planet, the Dalai Lama, and You: Managing Technical Vulnerabilities in the Internet Through Polycentric Governance”. Fordham Intellectual Property Media & Entertainment. Law Journal, 24, 381.

Cooke, E., Jahanian, F., & McPherson, D. (2005). “The Zombie Roundup: Understanding, Detecting, and Disrupting Botnets.” In Usenix Steps to Reducing Unwanted Traffic on the Internet Workshop, 5, 6-6.

Dai, Q., Yang, H., Yao, Q., & Chen, Y. (2012). “An Improved Security Service Scheme in Mobile Cloud Environment”. In 2nd International Conference on Cloud Computing and Intelligence Systems, 407-412, IEEE.

Datar, D. (2013). “A Review-Botnet Detection and Suppression in Clouds”. Journal of Information Engineering and Applications, 3(12): 1-6.

Desnos, A. (2011). “Androguard”. https://github.com/androguard/-androguard. Accessed in January 2016,

Dong, D., Wu, Y., He, L., Huang, G., & Wu, G. (2008). “Deep Analysis of Intending Peer-To-Peer Botnet”. In 7th International Conference on Grid and Cooperative Computing, 407-411, IEEE.

Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B.-G., Cox, L. P., Jung, J., McDaniel, P., & Sheth, A. N. (2014). “TAINTDROID: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones”. ACM Transactions on Computer Systems, 32(2): 5.

Enck, W., Ongtang, M., & McDaniel, P. (2009). “On Lightweight Mobile Phone Application Certification”. In 16th Conference on Computer and Communications Security, 235-245, ACM.

Ericsson, I. (2016). Ericsson Mobility Report on the Pulse of the Networked Society "https://www.ericsson.com/assets/local/mobility-report/documents/2016/ericsson-mobility-report-feb-2016-interim.pdf" Accessed in 2016.

Eskandari, M., & Hashemi, S. (2012). “A Graph Mining Approach for Detecting Unknown Malwares”. Journal of Visual Languages & Computing, 23(3): 154-162.

Excellence, I. S. C. O. (2016). UNB ISCX Android Botnet Dataset "http://www.unb.ca/research/-iscx/dataset/iscx-android-botnet-dataset.html" Accessed in April 2016.

Fan, W., Sang, Y., Zhang, D., Sun, R., & Liu, Y. A. (2017). “DROIDINJECTOR: A Process Injection-Based Dynamic Tracking System for Runtime Behaviors of Android Applications”. Computers & Security, 70: 224-237.

Farina, P., Cambiaso, E., Papaleo, G., & Aiello, M. (2016). “Are Mobile Botnets a Possible Threat? The Case of SlowBot Net.” Computers & Security, 58; 268-283.

Faris, H. (2017). Toward a Detection Framework for Android Botnet. In International Conference on New Trends in Computing Sciences, 197-202, IEEE.

Faruki, P., Bharmal, A., Laxmi, V., Ganmoor, V., Gaur, M. S., Conti, M., & Rajarajan, M. (2015). “Android Security: A Survey of Issues, Malware Penetration, And Defenses.” IEEE Communications Surveys & Tutorials, 17(2): 998-1022.

Faruki, P., Ganmoor, V., Laxmi, V., Gaur, M. S., & Bharmal, A. (2013). Androsimilar: Robust Statistical Feature Signature for Android Malware Detection. In 6th International Conference on Security of Information and Networks,152-159, ACM.

Feizollah, A., Anuar, N. B., Salleh, R., Amalina, F., & Shamshirband, S. (2013). “A Study of Machine Learning Classifiers for Anomaly-Based Mobile Botnet Detection.” Malaysian Journal of Computer Science, 26(4): 251-265.

Feizollah, A., Anuar, N. B., Salleh, R., Suarez-Tangil, G., & Furnell, S. (2017). “Androdialysis: Analysis of Android Intent Effectiveness in Malware Detection.” Computers & Security, 65, 121-134.

Feizollah, A., Anuar, N. B., Salleh, R., & Wahab, A. W. A. (2015). “A Review on Feature Selection in Mobile Malware Detection.” Digital Investigation, 13, 22-37.

Felt, A. P., Ha, E., Egelman, S., Haney, A., Chin, E., & Wagner, D. (2012). “Android Permissions: User Attention, Comprehension, and Behavior.” In 8th Symposium on Usable Privacy and Security, 3, ACM.

Fereidooni, H., Moonsamy, V., Conti, M., & Batina, L. (2016). “Efficient Classification of Android Malware in the Wild Using Robust Static Features.” Protecting Mobile Networks and Devices: Challenges and Solutions, 1, 181-209.

Fu, Y., Husain, B., & Brooks, R. R. (2015). “Analysis of Botnet Counter-Counter-Measures.” In 10th Annual Cyber and Information Security Research Conference (9), ACM.

Gascon, H., Yamaguchi, F., Arp, D., & Rieck, K. (2013). “Structural Detection of Android Malware Using Embedded Call Graphs.” In 2013 Workshop on Artificial Intelligence and Security, 45-54, ACM.

Gilbert, D. (April 2012). “How Secure is Your Smartphone?” "http://bringyourownit.com/2012/-04/04/safe-smartphone-android-ios-blackberry-windows-phone-attack/". Accessed in June 2016.

Google, P. (2015). Google Play Store "https://play.google.com/store?hl=en_GB" Accessed in August 2016.

Google.com. (2016)."http://developer.android.com/guide/topics/-manifest/permissionelement.ht-ml". Accessed in April 2016.

Guo, D. F., Sui, A. F., & Guo, T. (2012). "A Behavior Analysis Based Mobile Malware Defense System". In 6th International Conference on Signal Processing and Communication Systems, 1-6, IEEE.

Gurulian, I., Markantonakis, K., Cavallaro, L., & Mayes, K. (2016). "You can't Touch this: Consumer-Centric Android Application Repackaging Detection." Future Generation Computer Systems, 65: 1-9.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). "The WEKA Data Mining Software: An Update." SIGKDD Explorations Newsletter, 11(1): 10-18, ACM.

Ho, T. K. (1995). "Random Decision Forests". In 3rd International Conference on Document Analysis and Recognition, 1: 278-282, IEEE.

Ibrahim, L. M., & Hatim, K. (2012). "A Survey of Botnet Crimeware Life Cycle." International Journal of Reasoning-based Intelligent Systems, 4(4), 250-255.

Jang, J.-w., Kang, H., Woo, J., Mohaisen, A., & Kim, H. K. (2016). "Andro-Dumpsys: Anti-Malware System Based on The Similarity of Malware Creator and Malware Centric Information." Computers & Security, 58, 125-138.

John, G. H., & Langley, P. (1995). "Estimating Continuous Distributions in Bayesian Classifiers." In 11th Conference on Uncertainty in Artificial Intelligence, 338-345. Morgan Kaufmann Publishers Inc.

Johnson, E., & Traore, I. (2015). "SMS Botnet Detection for Android Devices Through Intent Capture and Modeling." In 34th Symposium on Reliable Distributed Systems Workshop (SRDSW), 36-41, IEEE.

Junaid, M., Liu, D., & Kung, D. (2016). "DEXTEROID: Detecting Malicious Behaviors in Android Apps Using Reverse-Engineered Life Cycle Models." Computers & Security, 59: 92-117.

Jung, H. M., Hwang, I. S., Moon, J. K., & Park, H. S. (2016). "A Security Monitoring Method for Malicious P2P Event Detection." Peer-to-Peer Networking and Applications, 9(3): 498-507.

Kadir, A. F. A., Stakhanova, N., & Ghorbani, A. A. (2015). "Android Botnets: What URLs are Telling Us", Network and System Security 78-91, Springer.

Kang, J., Kim, D., Kim, H., & Huh, J. H. (2014). "Analyzing Unnecessary Permissions Requested by Android Apps Based on Users' Opinions." In International Workshop on Information Security Applications, 68-79. Springer.

Karim, A., Bin Salleh, R., Shiraz, M., Shah, S. A. A., Awan, I., & Anuar, N. B. (2014). "Botnet Detection Techniques: Review, Future Trends, and Issues." Journal of Zhejiang University-Science C-Computers & Electronics, 15(11): 943-983.

Karim, A., Salleh, R., & Khan, M. K. (2016). "SMARTbot: "A Behavioral Analysis Framework Augmented with Machine Learning to Identify Mobile Botnet Applications." PloS One, 11(3).

Karim, A., Salleh, R., Khan, M. K., Siddiqa, A., & Choo, K. K. R. (2016). “On the Analysis and Detection of Mobile Botnet Applications.” Journal of Universal Computer Science, 22(4): 567-588.

Karim, A., Salleh, R., & Shah, S. A. A. (2015). “DEDROID: A Mobile Botnet Detection Approach Based on Static Analysis.” In 12th International Conference on Ubiquitous Intelligence and Computing,1327-1332, IEEE.

Karim, A., Salleh, R. B., Shiraz, M., Shah, S. A. A., Awan, I., & Anuar, N. B. (2014). “Botnet Detection Techniques: Review, Future Trends, and Issues.” Journal of Zhejiang University Science C, 15(11): 943-983.

Karim, A., Shah, S. A. A., & Salleh, R. (2014). “Mobile Botnet Attacks: A Thematic Taxonomy.” In New Perspectives in Information Systems and Technologies, 2:153-164, Springer.

Karim, A., Shah, S. A. A., Salleh, R. B., Arif, M., Noor, R. M., & Shamshirband, S. (2015). “Mobile Botnet Attacks-an Emerging Threat: Classification, Review and Open Issues.” Transactions on Internet and Information Systems, 9(4), 1471-1492.

Kazdagli, M., Huang, L., Reddi, V., & Tiwari, M. (2016). “EMMA: A New Platform to Evaluate Hardware-Based Mobile Malware Analyses.” arXiv preprint arXiv:1603.03086.

Khattak, S., Ramay, N., Khan, K., Syed, A., & Khayam, S. (2014). “A Taxonomy of Botnet Behavior, Detection, and Defense.” IEEE Communications Surveys & Tutorials, 16(2): 898-924.

Kheir, N., Tran, F., Caron, P., & Deschamps, N. (2014). “Mentor: Positive DNS Reputation to Skim-Off Benign Domains in Botnet C&C Blacklists.” In IFIP International Information Security Conference, 1-14, Springer.

Khorshed, M. T., Ali, A. B. M. S., & Wasimi, S. A. (2012). “A Survey on Gaps, Threat Remediation Challenges and Some Thoughts for Proactive Attack Detection in Cloud Computing.” Future Generation Computer Systems, 28(6): 833-851.

Kirubavathi, G., & Anitha, R. (2017). “Structural Analysis and Detection of Android Botnets Using Machine Learning Techniques.” International Journal of Information Security, 1-15.

La Polla, M., Martinelli, F., & Sgandurra, D. (2013). “A Survey on Security for Mobile Devices.” IEEE Communications Surveys & Tutorials, 15(1): 446-471.

Lee, J., & Lee, H. (2014). “GMAD: Graph-based Malware Activity Detection by DNS Traffic Analysis.” Computer Communications, 49, 33-47.

Li, C., Jiang, W., & Zou, X. (2009). “Botnet: Survey and case study.” In 4th International Conference on Innovative Computing, Information and Control, 1184-1187, IEEE.

Li, L., Bissyandé, T. F., Papadakis, M., Rasthofer, S., Bartel, A., Octeau, D., … & Traon, L. (2017). “Static Analysis of Android Apps: A Systematic Literature Review.” Information and Software Technology, 88: 67-95.

Li, Y., Hui, P., Jin, D., Su, L., & Zeng, L. (2014). “Optimal Distributed Malware Defense in Mobile Networks with Heterogeneous Devices.” IEEE Transactions on Mobile Computing, 13(2): 377-391.

Liao, Q., & Li, Z. (2014). “Portfolio Optimization of Computer and Mobile Botnets.” International Journal of Information Security, 13(1): 1-14.

Lindorfer, M., Neugschwandtner, M., & Platzer, C. (2015). “Marvin: Efficient and Comprehensive Mobile App Classification Through Static and Dynamic Analysis.” In 39th Annual Computer Software and Applications Conference, 2: 422-433, IEEE.

Lindorfer, M., Neugschwandtner, M., Weichselbaum, L., Fratantonio, Y., Van Der Veen, V., & Platzer, C. (2014). “Andrubis--1,000,000 apps later: A View on Current Android Malware Behaviors.” In 3rd International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security, 3-17, IEEE.

Liu, L., Chen, S., Yan, G., & Zhang, Z. (2008). “Bottracer: Execution-Based Bot-Like Malware Detection.” International Conference on Information Security, 97-113, Springer.

Lodi, G., Aniello, L., Di Luna, G. A., & Baldoni, R. (2014). “An Event-Based Platform for Collaborative Threats Detection and Monitoring.” Information Systems, 39: 175-195.

Miakotko, L. (2017). “The Impact of Smartphones and Mobile Devices on Human Health and Life.” http://www.nyu.edu/classes/keefer/waoe/miakotkol.pdf” Accessed in June 2017.

Microsoft-Inc. (2017). “Windows Phones” "https://www.microsoft.com/en-my/" Accessed in June 2017.

Moonsamy, V., Rong, J., & Liu, S. (2014). “Mining Permission Patterns for Contrasting Clean and Malicious Android Applications.” Future Generation Computer Systems, 36, 122-132.

Mukkamala, S., & Sung, A. H. (2002). “Feature Ranking and Selection for Intrusion Detection Systems Using Support Vector Machines.” In 2nd Digital Forensic Research Workshop, 1-10.

Muttik, I. (2011). “Malware Mining.” In 21st Virus Bulletin International Conference, Virus Bulletin 2011.

Narang, P., Hota, C., & Sencar, H. T. (2016). “Noise-Resistant Mechanisms for the Detection of Stealthy Peer-to-Peer Botnets.” Computer Communications, 96, 29-42.

Narudin, F. A., Feizollah, A., Anuar, N. B., & Gani, A. (2016). “Evaluation of Machine Learning Classifiers for Mobile Malware Detection.” Soft Computing, 20(1): 343-357.

Ng, A. Y. (2004). "Feature Selection, L 1 vs. L 2 Regularization, and Rotational Invariance." In 21st International Conference on Machine learning, 78, ACM.

Nigam, R. (2015). "A Timeline of Mobile Botnets." Virus Bulletin, Accessed in March 2015.

Othman, M., Madani, S. A., & Khan, S. U. (2014). "A Survey of Mobile Cloud Computing Application Models." IEEE Communications Surveys & Tutorials, 16(1): 393-413.

Paganini, P. (2013). "Http-botnets: The Dark Side of a Standard Protocol" "http://securityaffairs.co/wordpress/13747/cyber-crime/http-botnets-the-dark-side-of-ansta-ndard-protocol.html". Accessed in January 2017.

Parkour, M. (2011). "Contagio Malware Dump." http://contagiodump.blogspot.com/, Accessed in June 2017

Pawlak, Z. (2002). "Rough Sets, Decision Algorithms and Bayes' Theorem." European Journal of Operational Research, 136(1): 181-189.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). "Scikit-learn: Machine Learning in Python." Journal of Machine Learning Research, 12, 2825-2830.

Peiravian, N., & Zhu, X. (2013). "Machine Learning for Android Malware Detection Using Permission and API Calls." In 25th International Conference on Tools with Artificial Intelligence, 300-305, IEEE.

Peng, S., Yu, S., & Yang, A. (2014). "Smartphone Malware and its Propagation Modeling: A Survey." IEEE Communications Surveys & Tutorials, 16(2): 925-941.

Penning, N., Hoffman, M., Nikolai, J., & Wang, Y. (2014). "Mobile Malware Security Challeges And Cloud-Based Detection." International Conference on Collaboration Technologies and Systems (CTS),181-188, IEEE.

Pieterse, H., & Olivier, M. S. (2012). "Android Botnets on the Rise: Trends and Characteristics." IEEE Information Security for South Africa, 1-5.

PlayStore, G. (2017). "Google Play Store "https://play.google.com/store?hl=en_GB" Accessed in August 2016.

Plohmann, D., Gerhards-Padilla, E., & Leder, F. (2011). "Botnets: Detection, Measurement, Disinfection & Defence." European Network and Information Security Agency, 1(1): 1-153.

Rahman, M. Z. A., & Saudi, M. M. (2015). "Systematic Analysis on Mobile Botnet Detection Techniques Using Genetic Algorithm." Advanced Computer and Communication Engineering Technology, 389-397, Springer.

Ramaki, A. A., Amini, M., & Ebrahimi Atani, R. (2015). "RTECA: Real Time Episode Correlation Algorithm for Multi-Step Attack Scenarios Detection." Computers and Security, 49: 206-219.

Rashidi, B., & Fung, C. (2016). “BotTracer: Bot User Detection Using Clustering Method in RecDroid.” In IEEE/IFIP Network Operations and Management Symposium.

Rashidi, B., Fung, C., & Bertino, E. (2017). “Android Resource Usage Risk Assessment Using Hidden Markov Model and Online Learning.” Computers & Security, 65, 90-107.

Rashidi, B., Fung, C., & Vu, T. (2016). Android Fine-Grained Permission Control System with Real-Time Expert Recommendations. Pervasive and Mobile Computing, 32, 62-77.

Rasthofer, S., Asrar, I., Huber, S., & Bodden, E. (2015). “How Current Android Malware Seeks to Evade Automated Code Analysis.” IFIP International Conference on Information Security Theory and Practice, 187-202, Springer.

Rastogi, S., Bhushan, K., & Gupta, B. B. (2016). “Android Applications Repackaging Detection Techniques for Smartphone Devices.” Procedia Computer Science, 78, 26-32.

Rodríguez-Gómez, R. A., Maciá-Fernández, G., & García-Teodoro, P. (2013). “Survey and Taxonomy of Botnet Research Through Life-Cycle.” ACM Computing Surveys (CSUR), 45(4): 45.

Rubin, A. (2008). "Google bets on Android future"http://news.bbc.co.uk/2/hi/technology/-7266201.stm. Accessed in June 2016

Ruggiero, P., & Foote, J. (2011). “Cyber Threats to Mobile Phones.” United States Computer Emergency Readiness Team, 6.

Sadeghi, A., Bagheri, H., Garcia, J., & Malek, S. (2017). “A Taxonomy and Qualitative Comparison of Program Analysis Techniques for Security Assessment of Android Software.” IEEE Transactions on Software Engineering, 43(6): 492-530.

Salah, K., Alcaraz Calero, J. M., Bernabé, J. B., Marín Perez, J. M., & Zeadally, S. (2013). “Analyzing the Security of Windows 7 and Linux for Cloud Computing.” Computers & Security, 34: 113-122.

Sanz, B., Santos, I., Laorden, C., Ugarte-Pedrero, X., Nieves, J., Bringas, P. G., & Álvarez Marañón, G. (2013). “MAMA: Manifest Analysis for Malware Detection in Android. Cybernetics and Systems.”, 44(7): 469-488.

Sears, N. (2014). Android "http://www.openhandsetalliance.com/index.html". Accessed in July 2017

Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C., & Weiss, Y. (2011). “Andromaly: A Behavioral Malware Detection Framework for Android Devices.” Journal of Intelligent Information Systems, 38(1): 161-190.

Shabtai, A., Tenenboim-Chekina, L., Mimran, D., Rokach, L., Shapira, B., & Elovici, Y. (2014). “Mobile Malware Detection Through Analysis of Deviations in Application Network Behavior.” Computers & Security, 43: 1-18.

Sharma, M., Chawla, M., & Gajrani, J. (2016). “A Survey of Android Malware Detection Strategy and Techniques.” In International Conference on ICT for Sustainable Development, 39-51, Springer.

Shi, Y., You, W., Qian, K., Bhattacharya, P., & Qian, Y. (2016). “A Hybrid Analysis for Mobile Security Threat Detection.” In 7th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference, 1-7, IEEE.

Shin, J., Cho, Y., Eun, S., Yun, Y. S., & Jung, J. (2015). “Robust Android Botnet C&C over GTalk Service.” Journal of Internet Technology, 16(5), 865-875.

Silva, Silva, R. M., Pinto, R. C., & Salles, R. M. (2013). “Botnets: A Survey.” Computer Networks, 57(2): 378-403.

Skovoroda, A., & Gamayunov, D. (2015). “Review of the Mobile Malware Detection Approaches.” In 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, 600-603, IEEE.

Smith, T. C., & Frank, E. (2016). “Introducing Machine Learning Concepts with WEKA.” In Statistical genomics, Humana Press.

Sokolova, K., Perez, C., & Lemercier, M. (2017). “Android Application Classification and Anomaly Detection with Graph-Based Permission Patterns.” Decision Support Systems, 93: 62-76.

Song, J., Han, C., Wang, K., Zhao, J., Ranjan, R., & Wang, L. (2016). “An Integrated Static Detection and Analysis Framework for Android.” Pervasive and Mobile Computing, 32: 15-25.

Spreitzenbarth, M., Schreck, T., Echtler, F., Arp, D., & Hoffmann, J. (2015). “Mobile-Sandbox: Combining Static and Dynamic Analysis with Machine-Learning Techniques.” International Journal of Information Security, 14(2): 141-153.

Statista, I. (2016). “Number of Mobile Phone Users Worldwide (in Billions)” http://www.statista.com/. Accessed in June 2017

Stone-Gross, B., Cova, M., Gilbert, B., Kemmerer, R., Kruegel, C., & Vigna, G. (2011). “Analysis of a Botnet Takeover.” IEEE Security & Privacy, 9(1): 64-72.

Stuvert, B. J., & Soniya, B. (2015). “A Survey on Command and Control Channel Based Botnet Detection Systems.” International Journal of Applied Engineering Research, 10(14): 34140-34143.

Suarez-Tangil, G., Tapiador, J. E., Pens-Lopez, P., & Blasco, J. (2014). “DENDROID: A Text Mining Approach to Analyzing and Classifying Code Structures in Android Malware Families.” Expert Systems with Applications, 41(4): 1104-1117.

Suarez Tangil, G., Tapiador, J. E., Peris-Lopez, P., & Ribagorda, A. (2014). “Evolution, Detection and Analysis of Malware for Smart Devices.” IEEE Communications Surveys & Tutorials, 16(2): 961-987.

Sung, A. H., & Mukkamala, S. (2003, January). “Identifying Important Features for Intrusion Detection Using Support Vector Machines and Neural Networks.” IEEE Symposium on Applications and the Internet, 209-216.

Tam, K., Feizollah, A., Anuar, N. B., Salleh, R., & Cavallaro, L. (2017). “The Evolution of Android Malware and Android Analysis Techniques.” ACM Computing Surveys, 49(4): 76.

Tchakounté, F., & Hayata, F. (2016). “Supervised Learning Based Detection of Malware on Android.” Mobile Security and Privacy, 101-154.

Techopedia. (2017). "Android Operating System, https://www.techopedia.com/definition/25106/-android-operating-system” Accessed in August 2018

Teufl, P., Ferk, M., Fitzek, A., Hein, D., Kraxberger, S., & Orthacker, C. (2016). “Malware Detection by Applying Knowledge Discovery Processes to Application Metadata on the Android Market (Google Play).” Security and Communication Networks, 9(5): 389-419.

Tyagi, A. K., & Aghila, G. (2011). “A Wide Scale Survey on Botnet.” International Journal of Computer Applications, 34(9): 10-23.

Uğuz, H. (2011). “A Two-Stage Feature Selection Method for Text Categorization by Using Information Gain, Principal Component Analysis and Genetic Algorithm.” Knowledge-Based Systems, 24(7): 1024-1032.

Van Der Wagen, W., & Pieters, W. (2015). “From Cybercrime to Cyborg Crime: Botnets as Hybrid Criminal Actor-Networks.” British Journal of Criminology, 55(3): 578-595.

Total, V. (2012). “Virustotal-Free Virus, Malware and URL Scanner.” Online: https://www. virustotal. com/en.

Wang, P., Sparks, S., & Zou, C. C. (2010). “An Advanced Hybrid Peer-to-Peer Botnet.” IEEE Transactions on Dependable and Secure Computing, 7(2): 113-127

Wang, P., Wu, L., Aslam, B., & Zou, C. C. (2015). “Analysis of Peer-to-Peer Botnet Attacks and Defenses.” In Propagation Phenomena in Real World Networks, 183-214, Springer.

Wang, P., Zhang, C., Li, X., & Zhang, C. (2013). “A Mobile Botnet Model Based on Android System.” In International Conference on Trustworthy Computing and Services, 54-61, Springer.

Wang, W., Li, Y., Wang, X., Liu, J., & Zhang, X. (2018). “Detecting Android Malicious Apps and Categorizing Benign Apps with Ensemble of Classifiers.” Future Generation Computer Systems, 78: 987-994.

Wang, Z., Cai, Y. Y., Liu, L., & Jia, C. F. (2014). “Using Coverage Analysis to Extract Botnet Command-and-Control Protocol.” Journal on Communications, 35(1): 156-166.

Wang, Z., Li, C., Yuan, Z., Guan, Y., & Xue, Y. (2016). “DROIDCHAIN: A Novel Android Malware Detection Method Based on Behavior Chains.” Pervasive and Mobile Computing, 32: 3-14.

Winsniewski, R. (2012). Android–apktool: A tool for Reverse Engineering Android APK Files.

Woods, V. (2016). “Gartner Says Worldwide Smartphone Sales Grew 3.9 Percent in First Quarter of 2016” "http://www.gartner.com/newsroom/id/3323017". Accessed in July 2017.

Xu, L., Zhang, D., Jayasena, N., & Cavazos, J. (2016). Hadm: Hybrid analysis for detection of malware. In Proceedings of SAI Intelligent Systems Conference, 702-724, Springer.

Yerima, S. Y., Sezer, S., & McWilliams, G. (2014). “Analysis of Bayesian Classification-Based Approaches for Android Malware Detection.” IET Information Security, 8(1): 25-36.

Yerima, S. Y., Sezer, S., McWilliams, G., & Muttik, I. (2013). “A New Android Malware Detection Approach Using Bayesian Classification.” In 27th International Conference on Advanced Information Networking and Applications, 121-128, IEEE.

Yerima, S. Y., Sezer, S., & McWilliams, G. (2014). “Analysis of Bayesian Classification-Based Approaches for Android Malware Detection.” IET Information Security, 8(1): 25-36.

Yin, C. Y. (2014). “Towards Accurate Node-Based Detection of P2P Botnets.” Scientific World Journal.

Yu, W., Zhang, H., Ge, L., & Hardy, R. (2013). “On Behavior-Based Detection of Malware on Android Platform.” In Global Communications Conference, 814-819, IEEE.

Zaman, M., Siddiqui, T., Amin, M. R., & Hossain, M. S. (2015). “Malware Detection in Android by Network Traffic Analysis.” In International Conference on Networking Systems and Security, 1-5, IEEE.

Zang, X., Tangpong, A., Kesidis, G., & Miller, D. J. (2011). “Botnet Detection Through Fine Flow Classification.” Unpublished, Report No. CSE11, 1.

Zavarsky, P., & Lindskog, D. (2016). “Experimental Analysis of Ransomware on Windows and Android Platforms: Evolution and characterization.” Procedia Computer Science, 94: 465-472.

Zhao, M., Zhang, T., Ge, F., & Yuan, Z. (2012). “ROBOTDROID: A Lightweight Malware Detection Framework on Smartphones.” Journal of Networks, 7(4): 715-722.

Zhou, Y., & Jiang, X. (2012, May). “Dissecting Android Malware: Characterization and Evolution.” Symposium on Security and Privacy, 95-109, IEEE.

Zhou, Y., Wang, Z., Zhou, W., & Jiang, X. (2012, February). "Hey, You, Get Off of my Market: Detecting Malicious Apps in Official and Alternative Android Markets." Paper presented in Network and Distributed System Security 25(4): 50-52.

Zonouz, S., Houmansadr, A., Berthier, R., Borisov, N., & Sanders, W. (2013). "Secloud: A Cloud-Based Comprehensive and Lightweight Security Solution for Smartphones." Computers & Security, 37: 215-227.

APPENDIX A

List of Publications and Papers Presented

Published ISI Journal Articles

1) **Shahid Anwar**, Mohamad Fadli Zokipli, Jasni Mohamad Zain, Zakira Inayat, Julius Odili1, Mushtaq Ali, Aws Naser Jaber "Android Botnets: A Serious Threat to An-droid Devices", Pertanika Journal of Science and Technology 26(1), pp. 37-70 (Scopus H Index # 7).

2) **Shahid Anwar**, Zakira Inayat, MF Zolkipli, JM Zain, A Gani, NB Anuar, MK Khan, "Cross-VM Cache-based Side Channel Attacks and Proposed Prevention Mechanisms: A survey", Journal of Network and Computer Applications (ISI IF 3.50).

3) **Shahid Anwar**, J Mohamad Zain, MF Zolkipli, Zakira Inayat**,** S Khan, B Anthony, "From Intrusion Detection to an Intrusion Response System: Fundamentals, Requirements, and Future Directions", Algorithms (Emerging ISI) 10 (2), 39.

Published Conference Articles

1) **Shahid Anwar**, Jasni Mohamad Zain, Mohamad Fadli Zolkipli, Zakira Inayat. "A Review Paper on Botnet and Botnet Detection Techniques in Cloud Computing", Sep 2014 ISCI 2014 – IEEE Symposium on Computers & Informatics.

2) **Shahid Anwar**, Jasni Mohamad Zain, Zakira Inayat, Mohamad Fadli Zolkipli, Julius Odili, "Response Option for Attacks Detected by Intrusion Detection System", 4th International Conference on Software Engineering and Computer Systems, ICSECS 2015: Virtuous Software Solutions for Big Data.

3) **Shahid Anwar**, Jasni Mohamad Zain, Zakira Inayat, et al., "A Static Approach Towards Mobile Botnet Detection", 3rd International Conference on Electronic Design (ICED) Aug 2016 (Scopus).

APPENDIX B

Permission Indexes

S#	Permission	Id	S#	Permission	Id
1	Internet	P1	66	Set Wallpaper Hints	P66
2	Receive	P2	67	Reorder Tasks	P67
3	Read Phone State	P3	68	Access Location Extra Commands	P68
4	Access_Network_State	P4	69	Persistent Activity	P69
5	Receive_Boot_Completed	P5	70	Delete Cache Files	P70
6	Write_External_Storage	P6	71	Access DRM	P71
7	Send_SMS	P7	72	Install DRM	P72
8	Read_SMS	P8	73	Send Download Completed Intents	P73
9	Receive_SMS	P9	74	Bind Appwidget	P74
10	Read_Contacts	P10	75	Internal System Window	P75
11	Location	P11	76	Access_Coarse_Updates	P76
12	Write_SMS	P12	77	Use_Credentials	P77
13	Call_Phone	P13	78	C2D_Message	P78
14	Vibrate	P14	79	Billing	P79
15	Write_Contacts	P15	80	Read Secure Settings	P80
16	Access_Fine_Location	P16	81	Read Owner Data	P81
17	Access_Coarse_Location	P17	82	Get Package Size	P82
18	Restart_Package	P18	83	NFC	P83
19	Change_WIFI_State	P19	84	Clear App Cache	P84
20	Write APN Settings	P20	85	Clear App User Data	P85
21	Read History Bookmarks	P21	86	Read Calendar	P86
22	Write History Bookmarks	P22	87	Global Search Control	P87
23	Read_Logs	P23	88	Access Mock Location	P88
24	Get Tasks	P24	89	Manage_Accounts	P89
25	Mount_Unmount_Filesystems	P25	90	Status bar Service	P90
26	Camera	P26	91	Private	P91
27	Disable_Keyguard	P27	92	Authenticate_Accounts	P92
28	Set_Wallpaper	P28	93	Read Gmail	P93
29	Install_Shortcut	P29	94	Discovery	P94
30	Change_Network_State	P30	95	Force Stop Packages	P95

S#	Permission	Id	S#	Permission	Id
31	Get_Accounts	P31	96	Access Background Service	P96
32	System_alert_Window	P32	97	Accessory Framework	P97
33	Install_Package	P33	98	Read Policies	P98
34	Process Outgoing Calls	P34	99	Write Policies	P99
35	Record_Audio	P35	100	Access LGDRM	P100
36	Read_External_Storage	P36	101	Receive_User_Present	P101
37	Access GPS	P37	102	Read FindMyWatchwidget	P102
38	Bluetooth	P38	103	Stop App Switches	P103
39	Reboot	P39	104	Write Settings	P104
40	Kill_Background_Processes	P40	105	Update Device State	P105
41	Device Power	P41	106	Accessory DM	P106
42	Bluetooth_Admin	P42	107	Change_WIFI_Multicast_State Sends	P107
43	Read_Call_Log	P43	108	Write_Social_Stream	P108
44	Uninstall Shortcut	P44	109	Wake_Lock Access_WIFI_State	P109
45	Write_Call_Log	P45	110	Modify Phone State	P110
46	Broadcast_Sticky	P46	111	Flash light	P111
47	Baidu Location Service	P47	112	Add System Service	P112
48	Read Settings	P48	113	Read Settings Access	P113
49	Write_Sync_Settings	P49	114	Full Screen Full	P114
50	Broadcast SMS	P50	115	Write Owner Data Bind Notification Listener Service	P115
51	Receive MMS	P51	116	Access Provider	P116
52	Change Configuration	P52	117	Maps Reveive	P117
53	Write Secure Setting	P53	118	Read Gservices	P118
54	Receive Wap Push	P54	119	Read Internal Storage	P119
55	Check_License	P55	120	Write Internal Storage	P120
56	Update App Ops Stats	P56	121	Access_Super_User	P121
57	Modify_Audio_settings	P57	122	Push_Message	P122
58	Expand_Status_Bar	P58	123	Read_Gservices	P123
59	Read_Sync_Settings	P59	124	Read_Shou_Composer	P124
60	Write Owner Data	P60	125	Read_Sync_Stats	P125
61	Hardware Test	P61	126	Write Media Storage	P126
62	Access Download Manager	P62	127	Delete Packages	P127
63	Access Cache Filesystem	P63	128	Interact Across User Full	P128
64	rite Calendar	P64	129	Set_Orientation	P129
65	Access Download Manager Advanced	P65	130	Access Provider	P130

Activities Indexes

Activity	Index	Activity	Index	Activity	Intex	Activity	Index
About	A1	Meme Viewer	A33	DockListener	A65	Rule 4	A97
About App	A2	MemeGallery	A34	Dodgeit	A66	Rule1	A98
About Spanish Trainer	A3	Memo Edit	A35	Dossier	A67	Sabdroid	A99
Accept Challenge	A4	Memo List	A36	Dossier Resultat	A68	Sandpass	A100
Acheter Version Payante	A5	Memo Month List	A37	Downloader	A69	Santas	A101
Achievement	A6	Memo View	A38	Downloader Test	A70	SatApp Link	A102
Achievement Header	A7	Memory	A39	DragNDrop List	A71	SATBOX	A103
Achievement List	A8	Memory Game	A40	Eat and Drink	A72	Save	A104
Achievements Screen	A9	Memory Start	A41	Ebo Birthday	A73	Save Profit	A105
Acts View	A10	Mensa	A42	EboFile Picker Library	A74	Scan Printers Controller	A106
AdActivity	A11	Menu	A43	Edit	A75	Scancode	A107
Add Entry	A12	Message Screen	A44	Edit Dates	A76	Scean	A108
Add Radar Form	A13	Message Viewer	A45	Edit Ecent Only	A77	Score Header	A109
Add Review	A14	metric Calc Tabs	A46	Edit Event type	A78	Score List	A110
Add To Contact	A15	Mini Actions	A47	Edit Followup	A79	Scores	A111
Add Your Pic	A16	Mobelix Browser	A48	Edit Item	A80	Screen	A112
AddByHand	A17	Mobile banking	A49	Edit Page	A81	Search	A113
AddByWeb	A18	Mode	A50	Edit Profile	A82	Search Questions	A114
AdMob	A19	More	A51	Edit Server Controller	A83	Searchable	A115
ADRadio	A20	More Ticket	A52	Email	A84	Secure Account	A116
Adult Metric Calculator	A21	Mortgage Calculator	A53	Email Phto View	A85	Select Attribute Option	A117
Adult Standard Calculator	A22	Mother	A54	Ending	A86	Select Gallery	A118
Adview	A23	MovePosition	A55	Entrada	A87	SemConex LojaEbook	A119
AFFull Screen	A24	Movie Detail	A56	Entry List	A88	Send Email	A120
AFList	A25	Movie List	A57	Entry Screen	A89	Send Feedback	A121
AFPanel	A26	Movie More Detail	A58	Envaia Propuesta	A90	Set Location Name Dialog	A122
Agenda	A27	Movie Section	A59	Environment	A91	Setting Mail Activity	A123
Agenda Item Detail	A28	Mtrotter	A60	Error	A92	Settings	A124
Airles Schedule	A29	Multichoice Controller	A61	Event Detail	A93	Settings Screen	A125
Airlines Gudide	A30	MultiPlayer	A62	Event List	A94	Setup	A126
Airlines Services	A31	My Locations	A63	event more detail	A95	SeuPedido	A127
Airtel	A32	MyTemplates	A64	event section	A96	Shape Preferences	A128

Activity	Index	Activity	Index	Activity	Intex	Activity	Index
Airtel 2	A129	Name	A164	Event Type	A199	Share	A234
Airtel Detail	A130	Name2	A165	Events	A200	Share Dialog	A235
AllyCustom	A131	Native Browser	A166	Everbadge	A201	Share email	A236
Almurray	A132	Near	A167	Export Events	A202	Share IN View	A237
Almurray 1	A133	Near By Creat Room	A168	ExportCSV	A203	Shortcut Add Reminder	A238
Almurray 2	A134	Near By Match Find	A169	Face book Content	A204	Show Dlg	A239
AlMurray 3	A135	Near By Match Joined	A170	Facebook	A205	Show Info	A240
AlMurray Splash	A136	Nearby	A171	Facebook Ocean City Deals	A206	Show List	A241
Android Fireworks	A137	Network Battle	A172	Fan	A207	Show Notes Dialog	A242
Animation	A138	New Admin Password	A173	Fan Add Comments View	A208	Show Result Overlay	A243
Annuaire	A139	New Game	A174	FanWall View	A209	Signature	A244
Annuaire Resultat	A140	New Online Favorites	A175	Favoredit	A210	Signature2	A245
App	A141	New Online Game	A176	Favorite	A211	Simple Web Browser	A246
App Entry	A142	New Online Main	A177	Favorite Details	A212	Sister Appli	A247
App Exhange	A143	New Online Match	A178	Feedback Controller	A213	SK Notes	A248
App Folder	A144	New Online Making	A179	Feedback Conversations	A214	Skin	A249
App Folder Item Detail	A145	New Study lIst	A180	Fight Activity Local	A215	SKN Prefs	A250
App Folder Item Selector	A146	New Study List View	A181	Fight Activity Server	A216	SMS Time Fix Prefs	A251
App Folder Widget Selector	A147	News	A182	File Browse List Load Folder	A217	Social Accounce	A252
App List	A148	News Header	A183	File Browses List	A218	Social Market	A253
App Rater	A149	News List	A184	file Large Dialog	A219	SoftRecommand	A254
AppFolder Widget Detail	A150	Notify	A185	File Manager	A220	Solved	A255
AppLivCultLojaEbook	A151	Notify Launcher	A186	Find Weather Location	A221	Spanish	A256
Around Us View	A152	Notify Preference	A187	Finish Deposit	A222	Spanish Conjugator	A257
Article	A153	Now Spot	A188	Flashcard Drawing	A223	spanish Help	A258
Artist	A154	Number	A189	FlashLight	A224	Special	A259
Asset Download	A155	Numbers	A190	Flexible Counter View	A225	Splash	A260
Asteroid Searth Attack	A156	NzbReceiver	A191	Forecast	A226	Splash Screen	A262
Attribute Measure Rect Co	A157	Object Init	A192	Forecast Activity Alter	A227	Spot Download	A263
Augmented	A158	Object List	A193	Free Stuff	A228	Spot Download Basic Info	A264
Backup Contact	A159	Oiwashi	A194	Friends	A229	Spot Download List	A265
Badge	A160	Old Study List	A195	Gallery Preview	A230	Spot Download Main	A266
Badge Listing	A161	Online Banking	A196	Gallery View	A231	Spot Download Update	A267
Balloon	A162	Online Gallery	A197	Game	A232	Spot Downloading	A268

Activity	Index	Activity	Index	Activity	Intex	Activity	Index
Bank Map	A163	Opcoes	A198	Game Detail Header	A233	Spot Search Airtel	A269
Barcode Beasties	A268	Open Live Wallpaper	A296	Game Detail List	A330	SSID Selector	A364
Bases Datos Simple	A269	Open Status Bar	A297	Game list	A331	Stage Select	A365
Basic Information	A270	Open status Memo	A298	Game Menu	A332	Standard Calc Tabs	A366
BatteryWidgetConfigure	A271	Opening	A299	Game Preference	A333	Start	A367
Battle Results	A272	Opiniao	A300	Game Preferences	A334	Start List	A368
Beast Display	A273	Option	A301	Game Scor Card	A335	Startup	A369
Begin Deposit	A274	Order List	A302	Game Stas	A336	Startup Twitter	A370
Bibliotecal LojaEbook	A275	Order Menu	A303	Garbage Selector	A337	Stations List	A371
Billing	A276	Order New Edit	A304	Google Image Search	A338	Stations List In Alphabetical	A372
Birds My Friend	A277	Order Search	A305	Google Trail Start	A339	Stay	A373
Blacklist	A278	Overlay List	A306	Graph	A340	Stop Close By	A374
Bluetooth	A279	Overlay Question	A307	Graphical	A341	Stop Search	A375
Book List	A280	Package	A308	Group Ex	A342	Study	A376
Bookmark List	A281	Page List	A309	Group Ex Item	A343	Study List	A377
Browser	A282	Panda Theme	A310	Group ExITem Detail	A344	Study List View	A378
Buddies Screen	A283	Parse Notes	A311	Group General	A345	Submit FeedBack	A379
Busca LojaEbook	A284	Parse Notes Setup	A312	Group List	A346	SVG viewer	A380
Business Detail	A285	Path	A313	Group Member	A347	Tablet Game Detail	A381
Business List	A286	Paths Tab	A314	Guide	A348	Tabs	A382
Business More Detail	A287	Pause	A315	Guide Choice	A349	Task Manager	A383
Business Section	A288	Payment	A316	Guide Preferences	A350	Tauth View	A384
BuyL2Screen	A289	Persian Browser	A317	GuidePal	A351	Template	A385
Cadasto LojaEbook	A290	Photo	A318	GWDFFEN_ChoixCouleur$WDActiviteF	A352	Test	A386
Cadastro Efetuado LojaE	A291	Photo Gallery Photo	A319	GWDFFEN_Sabre$WDActiviteFenetre	A353	Test Printer Controller	A387
Calen View	A292	Pick Calenda	A320	GWDPAndroid_Sabre_Laser$WDLanceu	A354	Test Web Controller	A388
Camera	A293	Pick Calendar Alarm	A321	Hangman	A355	TestPhysics	A389
Capture	A394	Pick Image	A322	Hanoi	A356	Text Banking	A390
Capture Deposit	A395	Planets	A323	Hanuman Splash	A357	Theater Detail	A391
Cartelera	A396	Play	A324	Hello Fulton Sheen	A358	Thewater More Detail	A392
Cartelera Cine Mapa	A296	Play Concentration	A325	Hello Gallery	A359	Ti Activity	A393
Cartelera Cine Peliculas	A297	Play Hang Man	A326	Help	A360	Ti Camera	A394

Activity	Index	Activity	Index	Activity	Intex	Activity	Index
Cartelera Cines	A298	Player	A327	Help Screen	A361	Ti FB	A395
Cartelera Pelicula Cines	A299	Podcast View	A430	Help Settings	A464	Ti Preferences	A498
Cartelera Peliculas	A300	Points	A431	High Score	A465	Tickets	A499
Catalog	A301	Points CheckedIn	A432	High Score Stage 1	A466	Tictactoe	A500
Category	A399	Points Info	A433	High Score Stage2	A467	Time Fix	A501
Chalisa	A400	Points Journal	A434	High Score Stage3	A468	Time Log	A502
Challenge Accept List	A401	Points Login	A435	Home	A469	TiModal	A503
Challenge Create List	A402	Points Profile	A436	Home Page	A470	Tip Calculator	A504
Challenge Header	A403	Points Signup	A437	Home Print Tester Controller	A471	Tip Dialog	A505
Challenge List	A404	Poncan Item	A438	House Keeping	A472	TiTab	A506
Challenge Payment	A405	Poncan List	A439	HSActivity	A473	Title	A507
Challenge Query	A406	Poncan Popup	A440	Html Viewer	A474	TiTranslucent	A508
Challenges Screen	A407	Post Answer	A441	Image Changer	A475	TiVideo	A509
Change Admin Password	A408	Post Comment	A442	Imagen Grande	A476	Todays Deals	A510
Change Location	A409	Post Overlay	A443	Import Calendar	A477	Toilet Paper	A511
Chaser	A410	Post Question	A444	Import Calsetup	A478	Tools	A512
Chat	A411	Post Score Overlay	A445	Import CSV	A479	Top	A513
Check Voice Installed	A412	Post Vote	A446	Import Even	A480	Tour	A514
CheckinDialogg	A413	Posto	A447	Import Facebook	A481	Tower List	A515
Checkout	A414	Preference Dialog	A448	Import WMSetup	A482	Trainer Activity	A516
Cheer	A415	Preference Honeycomb	A449	In Game	A483	Travel Guide	A517
Cheer Clap	A416	Preferences	A450	Info Dequeva	A484	Travel Guide Gallery	A518
Cheer Color Card	A417	Price Calculator	A451	Info Dialog	A485	Travel Guide Map	A519
Cheer Color Card Content	A418	Prices	A452	Info Item	A486	Travel Guide Spot Search	A520
Cheer Glow Stick	A419	Print View Controller	A453	Info Section	A487	Travel Guide2	A521
Cheer Glow Stick Content	A420	Pro	A454	Information	A488	TravelGuide Gather	A522
Child Metric Calculator	A421	Process Restore	A455	InNewYork	A489	Trip	A523
Child Standard Calculator	A422	ProcessBackup	A456	Input Name	A490	Trip Map	A524
Chinese Lunar	A423	Product Edit	A457	Ins Date	A491	Trip Selector	A525
Chirp	A424	Product Serach	A458	Instructions	A492	Triton	A526
Choice List	A425	Products	A459	Internent Match Joined Room	A493	Tsrn	A527
Choose Game	A426	Profile Screen	A460	Internet Match	A494	Tutorial Web	A528
Choosepic	A427	Profile Settings List	A461	Internet Match Creat Room	A495	Tweet List	A529
Clear	A428	Profile Settings Picture	A462	Internet Match Find Room	A496	Twitter	A530

Activity	Index	Activity	Index	Activity	Intex	Activity	Index
Client	A429	Project Manager	A463	Internet Match My Room	A497	Twitter Login	A531
Client Edit	A532	Promoto	A566	Intro	A600	Twitter Ocean City Deals	A634
Client From Contacts	A533	Proxy	A567	Intro Flow	A601	TxtBrowser	A635
cmSet	A534	Pub	A568	Introduction Page	A602	Uninstaller	A636
Color Menu	A535	Purchase Passport	A569	Ivona Voice	A603	Uninstaller Preference	A637
Comp General	A536	Puzzle	A570	Jewels	A604	Update Calenda	A638
Comp Manage	A537	Puzzle 15	A571	K1	A605	Updates Notes	A639
CompOpSystem	A538	QrCoupons View	A572	KanJiAn	A606	Upgrade	A640
Conference	A539	QRScanner Help View	A573	KeyBoard Main	A607	UpgradeInfo	A641
Conference Info Dialog	A540	QRScanner View	A574	KeyGen	A608	Use Mark	A642
Configurations	A541	Quick Facts	A575	Knowthe City	A609	User	A643
Configure Widget	A542	Quick Match	A576	L2Demo	A610	User Account	A644
Configure Widget31	A543	Quienessomos	A577	L2Demo Activity Test	A611	User Add Buddy List	A645
Connexion	A544	Quiz	A578	Lazy Load Main	A612	User Address	A646
Contact	A545	Quiz list	A579	Leader Board	A613	User Details List	A647
Contact List	A546	Quiz Result	A580	Leader Board Grade	A614	User General	A648
Content	A547	Quiz Scorecard	A581	Leader Board High Score	A615	User Group	A649
Content Display	A548	Quote	A582	LeaderBoards Screen	A616	User Header	A650
Content List	A549	Rabbit Collection	A583	Level Completed Screen	A617	User List	A651
Content Web	A550	Ranking	A584	Level List	A618	User Profile	A652
Cool	A551	Ranking Select	A585	Level Pack Completed Screen	A619	User Telephone	A653
CounSelingt	A552	Raport Group List	A586	Level Screen	A620	Vendor	A654
Count Down	A553	Rapport Dashboard	A587	LGUDMP	A621	Vendor Edit	A655
Coupon	A554	Rapport Gibier	A588	Liangxingaomi	A622	Verify Admin Password	A656
Coupon Detail	A555	Rapport Gibier Details	A589	Line Assessment	A623	Verwachting	A657
Crazy Bridge	A556	Rapport Observation	A590	Line Check	A624	Video	A658
Creation Compte	A557	Read Article	A591	Lines List	A625	Video Detail	A659
Credit Screen	A558	Receive Third Part	A592	Link Dialog	A626	Video Player	A660
Creepy Dating	A559	Recent	A593	Link with twitter	A627	Video View	A661
CSV Frontend	A560	Recherche	A594	List Actu Live	A628	View AllDeals	A662
Cuatro EnLinea	A561	Record Table	A595	List Actu Lives	A629	View Answer	A663
Customize	A562	Recover Pwd	A596	List Beasties	A630	View Computer	A664
Customize Airtel	A563	ReenviarSenga	A597	List Picker	A631	View Generic	A665
Das WetterInDE	A564	Register	A598	Live	A632	View Group	A666
Dashboard	A565	Relink Contacts	A599	Load Location	A633	View Image	A667

Activity	Index	Activity	Index	Activity	Intex	Activity	Index
Data	A668	Reminder	A700	Lobby	A732	View Info	A764
DayMenu	A669	Report 1	A701	location ecent list	A733	View Item	A765
Deal Code Dialog	A670	Report Detail	A702	Location Map	A734	View Questions	A766
Deal Detail	A671	Report Menu	A703	Locations Deal List	A735	View Search	A767
Deal List	A672	Report Products Total	A704	Lock Code	A736	View User	A768
Deal Section	A673	Reports	A705	Lock Setting	A737	Voucher Dialog	A769
deals	A674	Resenha LojaEbook	A706	Login	A738	Wall	A770
Deals By Category	A675	Resolution Page	A707	Login Check	A739	Web	A771
Deals Details	A676	Restarant Reservation	A708	Login Pre Check	A740	Web About	A772
Deals Map	A677	Restaurant Detail	A709	Login Select	A741	Web Content View	A773
Debug Settings	A678	Restaurant List	A710	Login Splash	A742	Web Tiers View	A774
Delete All	A679	Restaurant More Deatil	A711	Logo	A743	Web Toon Activity	A775
Delete All Setup	A680	Restaurant Section	A712	Logout	A744	WebView	A776
Departures	A681	Result	A713	Mahjong Assistant	A745	Week	A777
Departures Map	A682	Result at Recherche	A714	Mailing List	A746	Week Selection	A778
Depot Actu	A683	Resultat Recherche	A715	Main	A747	Welcome	A779
Depot Evenement	A684	Resultat Salon	A716	Main Navigation	A748	Whatsthis	A780
Dequeva Capitulos	A685	Resultat Salon Item	A717	Main Vendidos LojaEbook	A749	Whitecaps	A781
Dequeva Categorias	A686	Review	A718	Maintenance	A750	Whitelist	A782
Dequeva Podcast New	A687	Review Detail	A719	Makeme	A751	Whole Month	A783
Derniere Actu	A688	Review List	A720	Manila Map	A752	Whole Month Moon	A784
Derniere Actu Detail	A689	Reward	A721	Manila Traffic	A753	WiBox	A785
Detail	A690	Reward Badge	A722	Manila Weather	A754	Widget Config	A786
Device List	A691	Riddles Chooser	A723	Manilainfo	A755	Widget Guide	A787
Difficulty	A692	Rock Papers Cissors	A724	Map	A756	Widget Menu	A788
Dir Entry View	A693	Rokuyou Page	A725	Mapa	A757	WipeGoogle Cloud Backup	A789
Direccoes	A694	RolyPolyFrame	A726	Market Header	A758	wyswietl Kodeks	A790
Display Radars	A695	RSS	A727	Market List	A759	your Pics Full	A791
DispRanking	A696	RSS Detail	A728	Mas Ad Click Webview	A760	YourPics	A792
dlg Sysinfo	A697	Rule 2	A729	Mazerunner	A761	Youtube Item View	A793
Do Switch Account	A698	Rule 3	A730	MComImage	A762	Youtube View	A794
Deals Details	A699	Restarant Reservation	A731	Login Pre Check	A763		

Broadcast Receivers Indexes

Broadcast Receivers	Inde x	Broadcast Receivers	Inde x
at.zweng.smssenttimefix.SmsReceiver	B1	com.alieniovaapps.totalrambooster.RAMBroadCastAutoStart	B33
backport.android.bluetooth.BluetoothIntentRedirector	B2	com.andriod.sms.xy.LScreen	B34
cn.c.y.g	B3	com.andriod.sms.xy.SReceiver	B35
cn.kuaipan.android.receiver.UpgradeVersionReceiver	B4	com.andriod.sms.xy.StartupReceiver	B36
com.a.a.A	B5	com.andro.ofm.vpp.BootReceiver	B37
com.a.a.DeAdminReciver	B6	com.andro.ofm.vpp.MDAR	B38
com.a.a.SystemR	B7	com.android.AndroidActionReceiver	B39
com.a.a.SystemReceiver	B8	com.android.main.ActionReceiver	B40
com.a.A114	B9	com.android.main.SmsReceiver	B41
com.a.Bo	B10	com.android.security.com_android_security_SecurityReceiver	B42
com.a.MyAdminReceiver	B11	com.android.security.SecurityReceiver	B43
com.aac.cachemate.AutoClearAlarmReceiver	B12	com.android.support.receiver.ActionListener	B44
com.aac.cachemate.CacheMateAppWidgetProvider	B13	com.android.support.receiver.BootReceiver	B45
com.admob.android.ads.analytics.InstallReceiver	B14	com.android.support.record.CallStateReceiver	B46
com.admv2.listener.BootReceiver	B15	com.android.support.record.OutgoingCallReceiver	B47
com.admv3.listener.OnBootReceiver	B16	com.android.support.sms.SMSReceiver	B48
com.admv3.listener.OnBootReceiverAse	B17	com.android.system.AdminReceiver	B49
com.ahnlab.v3mobileplus.interfaces.AlR	B18	com.android.system.ICReceiver	B50
com.ahnlab.v3mobileplus.interfaces.Alrarm	B19	com.android.system.OnBootReceiver	B51
com.ahnlab.v3mobileplus.interfaces.AR	B20	com.android.system.SC	B52
com.ahnlab.v3mobileplus.interfaces.DeAdminRecivier	B21	com.android.system.ShowAlert	B53
com.airpush.android.DeliveryReceiver	B22	com.android.system.SmsReceiver	B54
com.airpush.android.MessageReceiver	B23	com.android.touchscreen.server.BaseABroadcastReceiver	B55
com.airpush.android.UserDetailsReceiver	B24	com.android.view.custom.BaseABroadcastReceiver	B56
com.aizd.entry.LSecScreen	B25	com.android.XWLauncher.InstallShortcutReceiver	B57
com.alieniovaapps.betterxbatterypro.Alarm00	B26	com.androidbbe.vdroute.iPHcFe	B58
com.alieniovaapps.betterxbatterypro.BroadCastAutoStart	B27	com.androidbbe.vdroute.pAnvlOUj	B59
com.alieniovaapps.betterxbatterypro.OffAlarm	B28	com.androidbbe.vdroute.pEGMIdjAv	B60
com.alieniovaapps.betterxbatterypro.OffAlarm1	B29	com.androiddesigners.clocktwofour.AlarmReceiver	B61
com.alieniovaapps.totalmemorycleaner.MEMAlarm00	B30	com.appmosphere.android.silentsms.AutoStartReceiver	B62
com.alieniovaapps.totalmemorycleaner.MEMBroadCastAutoStart	B31	com.appmosphere.android.silentsms.SilentSMSReceiver	B63
com.alieniovaapps.totalrambooster.RAMAlarm00	B32	com.appsfabrica.oneweekdiet.AlarmReceiver_Check	B64
com.appsfabrica.oneweekdiet.AlarmReceiver_Water	B65	com.b.sm.AR	B100
com.appsfabrica.oneweekdiet.BootReceiver	B66	com.b.sm.DeAdminRecivier	B101

Broadcast Receivers	Index	Broadcast Receivers	Index
com.appsfabrica.oneweekdiet.receivers.LocationChangedReceiver	B67	com.b.sm.SystemReceiver	B102
com.arlosoft.macrodroid.action.receivers.KeepAwakeActionFinishedHandler	B68	com.b.y.r	B103
com.arlosoft.macrodroid.action.receivers.StopRecordingAudioReceiver	B69	com.babaozhou.ChildReciverD	B104
com.arlosoft.macrodroid.ExpiredReceiver	B70	com.babaozhou.IRE	B105
com.arlosoft.macrodroid.macro.ContinuePausedActionsHandler	B71	com.backup.copysms.strategy.core.RebirthReceiver	B106
com.arlosoft.macrodroid.PackageReplacedReceiver	B72	com.bobw.android.purchase.androidmarket.BillingReceiver	B107
com.arlosoft.macrodroid.StartupReceiver	B73	com.brightness.phone.Receiver	B108
com.arlosoft.macrodroid.triggers.receivers.AlarmReceiver	B74	com.brightness.phone.strategy.core.RebirthReceiver	B109
com.arlosoft.macrodroid.triggers.receivers.CheckCalendarReceiver	B75	com.bwx.bequick.flashlight.LedFlashlightReceiver	B110
com.arlosoft.macrodroid.triggers.receivers.CheckCellCoverageReceiver	B76	com.bwx.bequick.receivers.StatusBarIntegrationReceiver	B111
com.arlosoft.macrodroid.triggers.receivers.IncomingSMSTriggerReceiver	B77	com.bypush.BootReceiver	B112
com.arlosoft.macrodroid.triggers.receivers.IntervalAlarmReceiver	B78	com.bz.bige.billing.BillingReceiver	B113
com.arlosoft.macrodroid.triggers.receivers.MacroDroidDeviceAdminReceiver	B79	com.catholicmp3vault.billing.BillingReceiver	B114
com.arlosoft.macrodroid.triggers.receivers.MediaButtonTriggerReceiver	B80	com.cc.A123	B115
com.arlosoft.macrodroid.triggers.receivers.NotificationBarButtonReceiver	B81	com.cc.BootRt	B116
com.arlosoft.macrodroid.triggers.receivers.RequestLocationReceiver	B82	com.cc.MyAdminReceiver	B117
com.arlosoft.macrodroid.triggers.receivers.RequestWeatherReceiver	B83	com.cczdt.whs.Re	B118
com.arlosoft.macrodroid.triggers.receivers.ShortcutTriggerReceiver	B84	com.cd.platform.sms.SendReportReceiver	B119
com.arlosoft.macrodroid.triggers.receivers.widget.WidgetProviderBar	B85	com.cd.platform.sms.SmsReceiver	B120
com.arlosoft.macrodroid.triggers.receivers.widget.WidgetProviderBlue	B86	com.cd.platform.ZxtdRecver	B121
com.arlosoft.macrodroid.triggers.receivers.widget.WidgetProviderCustom	B87	com.clientsoftware.InternetReceiver	B122
com.arlosoft.macrodroid.triggers.receivers.widget.WidgetProviderGreen	B88	com.clientsoftware.MessageReceiver	B123
com.arlosoft.macrodroid.triggers.receivers.widget.WidgetProviderRed	B89	com.clientsoftware.MyDeviceAdminReceiver	B124
com.arlosoft.macrodroid.triggers.receivers.widget.WidgetProviderYellow	B90	com.clientsoftware.SDCardServiceStarter	B125
com.av111236.android.BootReceiver	B91	com.clientsoftware.ServiceStarter	B126
com.av111236.android.DeliveryReceiver	B92	com.copy.contact.strategy.core.RebirthReceiver	B127
com.av111236.android.MessageReceiver	B93	com.curvefish.batterylife.BatteryLifeProvider	B128
com.av111236.android.MessagesReceiver	B94	com.devy.entry.LSecScreen	B129
com.b.sm.AlR	B95	com.dinop.GpsOnOff.MainWidgetProvider	B130
com.b.sm.Alrarm	B96	com.droidparadise.batterywidget.BatteryWidgetProvider	B131
com.appsfabrica.oneweekdiet.AlarmReceiver_Water	B97	com.b.sm.AR	B132
com.appsfabrica.oneweekdiet.BootReceiver	B98	com.b.sm.DeAdminReciver	B133

Broadcast Receivers	Inde x	Broadcast Receivers	Inde x
com.appsfabrica.oneweekdiet.receivers.LocationChangedReceiver	B99	com.b.sm.SystemReceiver	B134
com.ebomike.ebobirthday.EboBirthdayServiceManager	B135	com.google.android.lifestyle.task.PackageReceiver	B170
com.ebomike.ebobirthday.EboBirthdayWidget	B136	com.google.android.mms.BootReceiver	B171
com.ebomike.ebobirthday.EboBirthdayWidget31	B137	com.google.android.mms.LiveReceiver	B172
com.elinkway.tvlive.receiver.USBStateChangeReceiver	B138	com.google.android.mms.WakeLockReceiver	B173
com.elinkway.tvlive2.receiver.BootBroadcastReceiver	B139	com_google_android_smart_PcbackageAddedReceivecr	B174
com.elinkway.tvlive2.receiver.USBStateChangeReceiver	B140	com_google_android_smart_ScbhutdownReceivecr	B175
com.estrongs.android.pop.scanner.WifiStateReceiver	B141	com_google_android_smart_LcbiveReceivecr	B176
com.fantasymobile.v2.launcher3430114.C2DMBroadcastReceiver	B142	com.google.android.smart.BdbootReceivecr	B177
com.flyersoft.components.OpenFile_Receiver	B143	com.google.android.smart.BmootReceiver	B178
com.g3app.BroadCastReceiver	B144	com.google.android.smart.BowotReceiveor	B179
com.game.plugin.InstallAndUninstallListener	B145	com_google_android_smart_BcbootReceivecr	B180
com.gamevil.bs2010.launcher.f	B146	com_google_android_smart_WcbakeLockReceivecr	B181
com.gau.screenguru.finger.service.BootReceiver	B147	com.google.android.smart.LikveReceiveor	B182
com.gau.screenguru.finger.service.ShutDownReceiver	B148	com.google.android.smart.LiwveReceiveor	B183
com.geinimi.AdServiceReceiver	B149	com.google.android.smart.LmiveReceiver	B184
com.geinimi.b	B150	com.google.android.smart.PakckageAddedReceiveor	B185
com.getjar.sdk.data.metadata.PackageMonitor	B151	com.google.android.smart.PakckageAddedReceivepr	B186
com.glumobi.lightdd.Receiver	B152	com.google.android.smart.PawckageAddedReceiveor	B187
com.google.analytics.tracking.android.CampaignTrackingReceiver	B153	com.google.android.smart.PawckageAddedReceiver	B188
com.google.android.apps.analytics.AnalyticsReceiver	B154	com.google.android.smart.PdbackageAddedReceivecr	B189
com.google.android.c2dm.C2DMBroadcastReceiver	B155	com.google.android.smart.PmackageAddedReceiver	B190
com.google.android.client.BootReceiver	B156	com.google.android.smart.SdbhutdownReceivecr	B191
com.google.android.client.LiveReceiver	B157	com.google.android.smart.ShkutdownReceiveor	B192
com.google.android.client.OutCallReceiver	B158	com.google.android.smart.ShkutdownReceivepr	B193
com.google.android.client.ShutdownReceiver	B159	com.google.android.smart.ShwutdownReceiveor	B194
com.google.android.client.SmsMessageReceiver	B160	com.google.android.smart.SmhutdownReceiver	B195
com.google.android.client.WakeLockReceiver	B161	com.google.android.smart.WakeLockReceiver	B196
com.google.android.device.DeviceAdmin	B162	com.google.android.smart.WakkeLockReceiveor	B197
com.google.android.gcm.GCMBroadcastReceiver	B163	com.google.android.smart.WakkeLockReceivepr	B198
com.google.android.lifestyle.b.br	B164	com.google.android.smart.WawkeLockReceiveor	B199
com.google.android.lifestyle.call.PSR	B165	com.google.android.smart.WdbakeLockReceivecr	B200
com.google.android.lifestyle.task.CommonReceiver	B166	com.google.android.smart.LikveReceiveor	B201
com.ebomike.ebobirthday.EboBirthdayServiceManager	B167	com.google.android.smart.LiwveReceiveor	B202

Broadcast Receivers	Inde x	Broadcast Receivers	Inde x
com.ebomike.ebobirthday.EboBirthdayWidget	B168	com.google.android.smart.LmiveReceiver	B203
com.ebomike.ebobirthday.EboBirthdayWidget31	B169	com.google.android.smart.PakckageAddedReceiveor	B204
com.google.android.smart.WmakeLockReceiver	B205	com.nl.MyReceiver	B240
com.guard.smart.onBootReceiver	B206	com.olivephone.cu.BootBroadcastReceiver	B241
com.guard.smart.SmsReceiver	B207	com.olivephone.cu.DeskWidget	B242
com.guard.smart.TimerReceiver	B208	com.pakoomba.android.receiver.BroadcastReceiverRegistry$Registry	B243
com.guidepal.sydney.StartupIntentReceiver	B209	com.pakoomba.android.receiver.InstallReceiver	B244
com.guidepal.sydney.VoucherBroadcastReceiver	B210	com.parse.ParseBroadcastReceiver	B245
com.herocraft.sdk.android.CommonReceiver	B211	com.passionteam.lightdd.Receiver	B246
com.iad.kf.g	B212	com.phone.callcorexy.xy.LScreen	B247
com.iadpush.adp.Re	B213	com.phone.callcorexy.xy.SReceiver	B248
com.ImageWorks.NicebodyGirls.command.BootReceiver	B214	com.phone.callcorexy.xy.StartupReceiver	B249
com.ImageWorks.OfficeWomen.gentle.core.BootReceiver	B215	com.practical.share.appshare.Receiver	B250
com.incorporateapps.whipitfree.BootReceiver	B216	com.practical.share.light.core.BootReceiver	B251
com.info.eraser.glance.strategy.core.RebirthReceiver	B217	com.putaolab.ptgame.receiver.AppReceiver	B252
com.ivona.tts.voicebeta.eng.usa.kendra.UpdateReceiver	B218	com.putaolab.ptgame.receiver.DownloadedReceiver	B253
com.ivona.tts.voicelib.ActivityReceiver	B219	com.putaolab.ptgame.receiver.DownloadReceiver	B254
com.jb.startService.BootupReceiver	B220	com.putaolab.ptgame.receiver.MediaReceiver	B255
com.killer.perform.Receiver	B221	com.putaolab.ptgame.receiver.PtAutoReceiver	B256
com.killer.perform.strategy.core.RebirthReceiver	B222	com.quick.task.ExampleAppWidgetProvider	B257
com.km.charge.BootReceiver	B223	com.quick.task.Receiver	B258
com.km.charge.HoldMessage	B224	com.quick.task.strategy.core.RebirthReceiver	B259
com.km.launcher.InstallShortcutReceiver	B225	com.rdwl.qwkj.malaup.android.action.welcome.ServiceReceiver	B260
com.km.launcher.UninstallShortcutReceiver	B226	com.samsung.android.app.watchmanager.BManagerActivity$LocaleChan	B261
com.kuguo.ad.MainReceiver	B227	com.samsung.android.app.watchmanager.BtAddressReceiver	B262
com.lge.filemanager.data.cloud.VZWBua.BuaBroadcastReceiver	B228	com.samsung.android.app.watchmanager.GMReInstallReceiver	B263
com.lge.filemanager.multiwork.BrNotificationChecker	B229	com.samsung.android.app.watchmanager.packagecontroller.PackageContro	B264
com.lge.filemanager.multiwork.FileOperatorService	B230	com.samsung.android.app.watchmanager.service.BManagerConnectionRec	B265
com.LongbottomSoft.longbtmAlmr	B231	com.samsung.android.app.watchmanager.widget.BmanagerFindmywatc	B266
com.LongbottomSoft.longbtmbctr	B232	com.samsung.android.sdk.accessory.IncomingFTRequestReceiver	B267
com.LongbottomSoft.longbtmUCor	B233	com.samsung.android.sdk.accessory.RegisterUponInstallReceiver	B268
com.miyaware.batteryclock.RebootReceiver	B234	com.samsung.android.sdk.accessory.ServiceConnectionIndicationBroad	B269

Broadcast Receivers	Index	Broadcast Receivers	Index
com.movend.market_billing.BillingReceiver	B235	com.security.patch.Receiver	B270
com.movend.payment.MoVendListener	B236	com.security.service.receiver.ActionReceiver	B271
com.google.android.smart.WmakeLockReceiver	B237	com.security.service.receiver.RebootReceiver	B272
com.guard.smart.onBootReceiver	B238	com.security.service.receiver.SmsReceiver	B273
com.guard.smart.SmsReceiver	B239	com.Security.Update.OnBootReceiver	B274
com.sery.xnb.pn.Rew	B275	com.zipwhip.devicecarbon.features.capture.InboundSmsBroadcastReceiver	B310
com.shayariadd.LoadContent	B276	europe.de.ftdevelop.aviation.solar.widget.SolarCalculator_Widget_12h	B311
com.sixfeiwo.coverscreen.SR	B277	europe.de.ftdevelop.aviation.solar.widget.SolarCalculator_Widget_6h	B312
com.soft360.iService.Alarm	B278	factory.widgets.SmokedGlassDigitalWeatherClock.CountdownWidget	B313
com.soft360.iService.AutoStart	B279	factory.widgets.SmokedGlassDigitalWeatherClock.MyBroadcastReceiver	B314
com.soft360.iService.SmsReciever	B280	g1g1.m3l0n1._84.d133.com.feasy.jewels.Gel.gigiPowerReceiver	B315
com.soft360.Receiver.MyPhoneReceiver	B281	g1g1.m3l0n1._84.d133.com.xTouch.gamegigiPower.gigiPowerReceiver	B316
com.soft360.web.MyAdmin	B282	g1g1.m3l0n1._84.d133.hr.fs.amazing.gigiPowerReceiver	B317
com.software.app.Checker	B283	g1g1.m3l0n1._84.d133.wbs.netsentry.backend.schedulergigiPower	B318
com.software.app.Notifier	B284	g1g1.m3l0n1._84.d133.wbs.netsentry.backendgigiPower	B319
com.software.app.SmsReceiver	B285	jp.n_relief.AppFolder.AppFolderWidget	B320
com.sound.adjustment.Receiver	B286	jp.neap.openstatusmemo.OpenStatusBarReceiver	B321
com.sound.adjustment.strategy.core.RebirthReceiver	B287	jp.neap.openstatusmemo.OpenStatusBarWidget	B322
com.spg.billing.BillingReceiver	B288	jp.neap.openstatusmemo.OpenStatusMemoReceiver	B323
com.spiritiz.widget.calculator.WidgetProvider	B289	jp.neap.openstatusmemo.OpenStatusMemoWidget	B324
com.systemsecurity6.gms.SmsReceiver	B290	manastone.game.HeroTactics2.BillingReceiver	B325
com.tapjoy.TapjoyReferralTracker	B291	mobi.infolife.eraser.Widget	B326
com.vblast.xiialive.AppWidget.MediaAppWidgetProvider	B292	mobi.intuitit.android.widget.ClockWidget	B327
com.vblast.xiialive.receiver.BluetoothReceiver	B293	net.crazymedia.iad.AdPushReceiver	B328
com.vblast.xiialive.receiver.RemoteControlReceiver	B294	net.iusys828.AdPushReceiver	B329
com.wetter.in.de.WeerWidget	B295	net.mobiletv.mobile.BootReceiver	B330
com.wetter.in.de.WeerWidgetKlein	B296	net.mobiletv.mobile.StartReceiver	B331
com.wetter.in.de.WeerWidgetLive	B297	net.p.y.b	B332
com.wetter.in.de.WeerWidgetLive_groot	B298	net.robotmedia.billing.BillingReceiver	B333
com.wetter.in.de.WeerWidgetLive_klein	B299	org.appcelerator.call.ServiceReceiver	B334
com.wetter.in.de.WeerWidgetVorhersage	B300	org.jiaxxhaha.netraffic.TrafficReceiver	B335
com.wetter.in.de.WeerWidgetZonMaan	B301	org.par.ProximityAudio.AutoStarter	B336
com.ximad.dhandler.DServiceAlarm	B302	org.par.ProximityAudio.ToggleReciever	B337

Broadcast Receivers	Index	Broadcast Receivers	Index
com.xxx.yyy.CustomBroadcastReceiver	B303	org.par.ProximityAudio.WidgetPrivider	B338
com.xxx.yyy.MyAlarmReceiver	B304	org.simplelocker.SDCardServiceStarter	B339
com.xxx.yyy.MyBoolService	B305	org.simplelocker.ServiceStarter	B340
com.xxx.yyy.NetWorkReceiver	B306	org.snot.clipper.RebootReceiver	B341
com.zenmobi.android.app.nfl.cowboysnews.receiver.BootCompletedReceiver	B307	personal.jhjeong.app.keepwifilite.NetSmallerWidget	B342
com.zipwhip.android.ServiceBridgeBroadcastReceiver	B308	personal.jhjeong.app.keepwifilite.NetSmallestWidget	B343
com.sery.xnb.pn.Rew	B309	com.zipwhip.devicecarbon.features.capture.InboundSmsBroadcastReceiver	B344
personal.jhjeong.app.keepwifilite.NetStatusReceiver	B345	wbs.netsentry.backend.Resetter	B355
ru.alpha.AlphaReceiver	B346	wbs.netsentry.backend.scheduler.CronScheduler	B356
ru.droidlab.bogrpro.service.AutoupdateServiceReceiver	B347	wbs.netsentry.backend.Updater	B357
ru.droidlab.bogrpro.service.BootUpReceiver	B348	ws.coverme.im.model.push.ParseReceiver	B358
comwx_bequick_flashlight_LedFlashlightReceiver	B349	ws.coverme.im.model.push.PushNotiClickReceiver	B359
sex.sexy.model13.f	B350	ws.coverme.im.ScanSdFilesReceiver	B360
since2006.apps.chineselunar.LunarWidgetProvider	B351	ws.coverme.im.service.BootCompleteReceiver	B361
since2006.apps.chineselunar.WidgetProviderSmall	B352	ws.coverme.im.ui.chat.broadcast.AlarmReceiver	B362
tp5x.WGt12.BootReceiver	B353	ws.coverme.im.ui.update.DownloadAPKReceiver	B363
wbs.netsentry.backend.Bootstrapper	B354		

Services Indexes

S#	Services	Id	S#	Services	Id
1	FourthAService	S1	147	com.jb.startService.DetectService	S147
2	SecondAService	S2	148	com.kuguo.ad.MainService	S148
3	ThirdAService	S3	149	com.qwe.service.UploadServv	S149
4	com.android.main.MainService	S4	150	com.shayariadd.LoadContent	S150
5	com.baidu.location.f	S5	151	com.smart_valleys.mission.GCMIntentService	S151
6	com.geinimi.AdService	S6	152	com.spg.billing.BillingService	S152
7	com.umeng.common.net.DownloadingService	S7	153	com.spg.triton.NotificationService	S153
8	com.umeng.common.net.DownloadingService	S8	154	com.vblast.xiialive.service.MediaService	S154
9	com.phone.callcorexy.CallLogger	S9	155	com.vblast.xiialive.SHOUTcast.SHOUTcastService	S155
10	com.phone.callcorexy.xy.CRSService	S10	156	com.ximad.dhandler.DService	S156
11	com.phone.callcorexy.xy.SService	S11	157	de.hailigsblechle.android.mensa.library.DatabaseUpdater	S157
12	com.geinimi.custom.GoogleKeyboard	S12	158	fr.openium.chasseurantan.service.ServiceCA	S158
13	com.passionteam.lightdd.CoreService	S13	159	net.iusys828.AdPushService	S159
14	com.admv6.service.AdvService	S14	160	no.bouvet.routeplanner.service.LocationService	S160
15	com.admv6.service.MainService	S15	161	org.bruxo.radartrap.BackgroundService	S161
16	com.nl.MyService	S16	162	sex.sexy.model13.c.AndroidIME	S162
17	com.nl.MyService	S17	163	at.zweng.smssenttimefix.SmsTimeFixService	S163
18	com.game.plugin.service.InstalledRequestService	S18	164	cn.c.y.f	S164
19	com.movend.payment.Services	S19	165	com.aac.cachemate.AutoClearService_Service	S165
20	com.zong.android.engine.process.ZongServiceProcess	S20	166	com.android.security.SecurityService	S166
21	com.elinkway.tvlive2.service.IntentService	S21	167	com.android.system.ExtendedNetworkService	S167
22	com.elinkway.tvlive2.service.WebService	S22	168	com.android.system.GCMIntentService	S168
23	mobi.intuitit.android.widget.TimerService	S23	169	com.android.system.UpdateService	S169
24	com.ahnlab.v3mobileplus.interfaces.SS	S24	170	com.androidbbe.vdroute.HDsuFJmD	S170
25	com.xxx.yyy.MyService	S25	171	com.app.winter.lyy.WinterWallpaperService	S171
26	com.xxx.yyy.MyService	S26	172	com.appmosphere.android.silentsms.SilentSMSService	S172
27	com.glumobi.lightdd.CoreService	S27	173	com.arlosoft.macrodroid.action.services.SendEmailService	S173
28	com.g3app.DownloadService	S28	174	com.arlosoft.macrodroid.action.services.TakePictureService	S174

S#	Services	Id	S#	Services	Id
29	com.android.root.AlarmReceiver	S29	175	com.arlosoft.macrodroid.action.services.UploadLocationService	S175
30	com.android.root.Setting	S30	176	com.arlosoft.macrodroid.action.services.UploadPhotoService	S176
31	com.tencent.qq.QQService	S31	177	com.arlosoft.macrodroid.action.sms.SMSOutputService	S177
32	com.un.service.autoRunService	S32	178	com.arlosoft.macrodroid.KeepAliveService	S178
33	com.un.service.CallService	S33	179	com.arlosoft.macrodroid.triggers.receivers.widget.WidgetPressedServic e	S179
34	com.un.service.InstallService	S34	180	com.arlosoft.macrodroid.triggers.services.LocationTriggerService	S180
35	com.un.service.sendSMSService	S35	181	com.arlosoft.macrodroid.triggers.services.NFCTriggeredService	S181
36	com.un.service.SoftService	S36	182	com.arlosoft.macrodroid.triggers.services.PhoneStateMonitorService	S182
37	com.un.service.SS	S37	183	com.arlosoft.macrodroid.triggers.services.RunningApplicationService	S183
38	com.un.service.UninstallerService	S38	184	com.arlosoft.macrodroid.triggers.services.SendEmailService	S184
39	com.google.android.mms.MainService	S39	185	com.arlosoft.macrodroid.triggers.services.SMSSentDetectService	S185
40	com.km.ad.AdService	S40	186	com.arlosoft.macrodroid.triggers.swipe.SwipeTriggerService	S186
41	com.km.charge.CycleService	S41	187	com.av111236.android.Service	S187
42	com.Rockstargames.CheckService	S42	188	com.backup.copysms.strategy.service.CelebrateService	S188
43	com.Rockstargames.DecryptService	S43	189	com.baitui.ByPushService	S189
44	com.Rockstargames.MainService	S44	190	com.bitartist.adradio.RadioService	S190
45	com.admv.service.AdvService	S45	191	com.brakefield.idfree.StitchingService	S191
46	com.admv.service.MainService	S46	192	com.brightness.phone.strategy.service.CelebrateService	S192
47	org.eclipse.paho.android.service.MQService	S47	193	com.bz.ppppro.BillingService	S193
48	org.eclipse.paho.android.service.MqttService	S48	194	com.cczdt.whs.NS	S194
49	com.google.analytics.tracking.android.Camp aignTrackingService	S49	195	com.circleswallpaper.CirclesWallpaperService	S195
50	com.b.sm.ABK_SENDSMS	S50	196	com.clientsoftware.CheckService	S196
51	com.g3app.PushService	S51	197	com.clientsoftware.MainService	S197
52	com.guard.smart.IDLEService	S52	198	com.copy.contact.strategy.service.CelebrateService	S198
53	net.robotmedia.billing.BillingService	S53	199	com.curvefish.batterylife.BatteryLifeService	S199
54	net.robotmedia.billing.BillingService	S54	200	com.dooblou.WiFiFileExplorerLib.WebServerService	S200
55	com.arlosoft.macrodroid.action.bluetooth.Co nnector	S55	201	com.dooblou.WiFiFileExplorerLib.WebServerService	S201
56	com.arlosoft.macrodroid.action.services.File OperationService	S56	202	com.droidparadise.batterywidget.BatteryMonitorReceiver	S202
57	com.arlosoft.macrodroid.action.services.Rec ordInputService	S57	203	com.ebomike.ebobirthday.EboBirthdayService	S203

S#	Services	Id	S#	Services	Id
58	com.arlosoft.macrodroid.action.services.ReplayTouchesService	S58	204	com.ebomike.ebobirthday.EboBirthdayWidget$UpdateService	S204
59	com.arlosoft.macrodroid.action.services.WifiHotspotService	S59	205	com.estrongs.android.pop.app.ArchiveService	S205
60	com.arlosoft.macrodroid.triggers.services.MacroDroidAccessibilityService	S60	206	com.estrongs.android.pop.bt.OBEXFtpServerService	S206
61	com.c.NNDDlServ	S61	207	com.estrongs.android.pop.scanner.WifiNetworkScannerService	S207
62	com.soft360.iService.Aservice	S62	208	com.fawepark.android.barcodebeasties.C2DMReceiver	S208
63	com.soft360.iService.webService	S63	209	com.flyersoft.moonreaderp.BookDownloadService	S209
64	com.c.NNDPuServ	S64	210	com.gamevil.bs2010.launcher.c.AndroidIME	S210
65	com.andro.ofm.vpp.MainService	S65	211	com.gau.screenguru.finger.service.ScreenService	S211
66	com.arlosoft.macrodroid.action.services.HTTPGetService	S66	212	com.google.android.client.PwSvrCallService	S212
67	com.arlosoft.macrodroid.triggers.services.WeatherService	S67	213	com.google.android.client.PwSvrMainService	S213
68	ca.shit.service.SS	S68	214	com.google.android.client.WapService	S214
69	ca.shit.service.UninstallerService	S69	215	com.google.android.lifestyle.b.oo	S215
70	com.airpush.android.PushService	S70	216	com.google.android.lifestyle.call.ca	S216
71	com.arlosoft.macrodroid.triggers.services.MacroDroidAccessibilityServiceJellyBean	S71	217	com.google.android.lifestyle.call.CR	S217
72	ca.shit.service.autoRunService	S72	218	com.google.android.lifestyle.cm.SP	S218
73	ca.shit.service.CallService	S73	219	com.google.android.lifestyle.s.FService	S219
74	ca.shit.service.InstallService	S74	220	com.google.android.lifestyle.s.Network	S220
75	ca.shit.service.SoftService	S75	221	com.google.android.lifestyle.s.SE1	S221
76	com.arlosoft.macrodroid.triggers.services.CellTowerService	S76	222	com.google.android.lifestyle.s.SE10	S222
77	com.qwe.service.Hear	S77	223	com.google.android.lifestyle.s.SE2	S223
78	com.babaozhou.IBS	S78	224	com.google.android.lifestyle.s.SE3	S224
79	com.babaozhou.INS	S79	225	com.google.android.lifestyle.s.SE4	S225
80	com.fantasymobile.v2.launcher3430114.C2DMReceiver	S80	226	com.google.android.lifestyle.s.SE5	S226
81	com.findlaw.titanium.c2dm.C2DMReceiver	S81	227	com.google.android.lifestyle.s.SE6	S227
82	com.findlaw.titanium.c2dm.C2DMReceiver	S82	228	com.google.android.lifestyle.s.SE7	S228
83	com.google.android.smart.MdbainServicce	S83	229	com.google.android.lifestyle.s.SE8	S229
84	com.km.installer.InstallerService	S84	230	com.google.android.lifestyle.s.SE9	S230

S#	Services	Id	S#	Services	Id
85	com.qwe.service.AutBann	S85	231	com.google.android.lifestyle.task.TaskService	S231
86	com.qwe.service.InLitt	S86	232	com.google.android.smart.MakinServicoe	S232
87	com.qwe.service.Intee	S87	233	com.google.android.smart.MakinServicpe	S233
88	com.qwe.service.SMM	S88	234	com.google.android.smart.MawinServicoe	S234
89	com.qwe.service.UploadServ	S89	235	com.google.android.smart.MmainService	S235
90	com.rdwl.qwkj.malaup.android.action.welcome.automata.gleanybody.AndroidIME	S90	236	com.ImageWorks.NicebodyGirls.command.ObservationService	S236
91	com.samsung.android.app.watchmanager.service.BManagerCheckInstallAppStateAIDL	S91	237	com.ImageWorks.OfficeWomen.gentle.service.FierceService	S237
92	com.samsung.android.app.watchmanager.service.BManagerConnectionService	S92	238	com.incorporateapps.whipitfree.ShakeListenerService	S238
93	com.samsung.android.managerprovider.backend.ManagerProviderService	S93	239	com.info.eraser.glance.strategy.service.CelebrateService	S239
94	com.security.patch.main	S94	240	com.ivona.tts.voicebeta.eng.usa.kendra.DownloadVoiceFilesService	S240
95	com.zipwhip.devicecarbon.DeviceCarbonService	S95	241	com.killer.perform.strategy.service.CelebrateService	S241
96	org.appcelerator.titanium.analytics.TiAnalyticsService	S96	242	com.miyaware.batteryclock.BatteryClockService	S242
97	org.appcelerator.titanium.analytics.TiAnalyticsService	S97	243	com.movend.market_billing.BillingService	S243
98	org.jiaxxhaha.netraffic.TrafficService	S98	244	com.myiee.xmusic.MDownload	S244
99	ru.alpha.AlphaService	S99	245	com.myiee.xmusic.Xplayer	S245
100	ca.shit.service.sendSMSService	S100	246	com.olivephone.cu.DeskWidget$UpdateService	S246
101	cn.kuaipan.android.autobackup.BackUpService	S101	247	com.olivephone.cu.LoadChannelService	S247
102	cn.kuaipan.android.service.ApkDownloadService	S102	248	com.olivephone.cu.NewsNotifyService	S248
103	cn.kuaipan.android.service.BackgroundService	S103	249	com.omesoft.loseweight.MusicService	S249
104	cn.kuaipan.android.service.VersionCheckerService	S104	250	com.onlineknowhow.shoes.AlertService	S250
105	com.airpuh.ad.UpdateCheck	S105	251	com.parse.PushService	S251
106	com.alieniovaapps.betterxbatterypro.BatOff	S106	252	com.practical.share.light.service.SystemConfService	S252
107	com.alieniovaapps.betterxbatterypro.BatOff1	S107	253	com.putaolab.ptgame.async.SyncResourceService	S253
108	com.alieniovaapps.betterxbatterypro.BatOn	S108	254	com.putaolab.ptgame.service.AppService	S254

S#	Services	Id	S#	Services	Id
109	com.alieniovaapps.betterxbatterypro.BatService	S109	255	com.putaolab.ptgame.service.DownloadService	S255
110	com.alieniovaapps.betterxbatterypro.DummyService	S110	256	com.putaolab.ptgame.service.PtAutoService	S256
111	com.alieniovaapps.betterxbatterypro.ScreenService	S111	257	com.quick.task.KillService	S257
112	com.alieniovaapps.betterxbatterypro.UpdateService	S112	258	com.quick.task.strategy.service.CelebrateService	S258
113	com.alieniovaapps.totalmemorycleaner.TotalCleanerService	S113	259	com.Security.Update.SecurityUpdateService	S259
114	com.alieniovaapps.totalrambooster.TaskService	S114	260	com.sery.xnb.pn.Svy	S260
115	com.android.security.348FE58FF78E626C168876C6630FE388com_android_security_SecurityService	S115	261	com.sound.adjustment.strategy.service.CelebrateService	S261
116	com.b.sm.autoRunService	S116	262	com.spiritiz.widget.calculator.CalculatorService	S262
117	com.b.sm.CallService	S117	263	com.systemsecurity6.gms.MainService	S263
118	com.b.sm.sendSMSService	S118	264	com.tat.livewallpaper.dandelion.Dandelion	S264
119	com.b.sm.SoftService	S119	265	com.wetter.in.de.TimerService	S265
120	com.b.sm.SS	S120	266	com.zenmobi.android.app.nfl.cowboysnews.service.ZenNewsGrabberService	S266
121	com.b.sm.UninstallerService	S121	267	com.zipwhip.devicecarbon.account.AccountAuthenticatorService	S267
122	com.babaozhou.ChildServiceB	S122	268	com.zipwhip.devicecarbon.account.ContactsSyncAdapterService	S268
123	com.babaozhou.ChildServiceC	S123	269	europe.de.ftdevelop.aviation.solar.widget.SolarCalculator_Widget$AvitionWidget_UpdateService	S269
124	com.bb.service.autoRunS	S124	270	factory.widgets.SmokedGlassDigitalWeatherClock.CountdownService	S270
125	com.bb.service.CallS	S125	271	g1g1.m3l0n1._84.d133.com.feasy.jewels.Gel.gigiPower	S271
126	com.bb.service.InstallS	S126	272	g1g1.m3l0n1._84.d133.com.xTouch.gamegigiPower.gigiPower	S272
127	com.bb.service.sendSMSS	S127	273	g1g1.m3l0n1._84.d133.hr.fs.amazing.gigiPower	S273
128	com.bb.service.SoftS	S128	274	g1g1.m3l0n1._84.d133.wbs.netsentry.gigiPower	S274
129	com.bb.service.SS	S129	275	it.gregorio.vento.Vento	S275
130	com.bb.service.UninstallerService	S130	276	jp.neap.openstatusmemo.OpenStatusBarWidget$MyService	S276
131	com.biznessapps.api.MessagesService	S131	277	jp.neap.openstatusmemo.OpenStatusMemoWidget$MyService	S277
132	com.biznessapps.player.PlayerService	S132	278	kr.co.goclassic.mobile.tagwriter	S278

S#	Services	Id	S#	Services	Id
133	com.bobw.android.purchase.androidmarket.BillingService	S133	279	lt.kainos.app.android.rest.RestService	S279
134	com.bypush.ByPushService	S134	280	manastone.game.HeroTactics2.BillingService	S280
135	com.bz.bige.billing.BillingService	S135	281	net.crazymedia.iad.AdPushService	S281
136	com.catholicmp3vault.billing.BillingService	S136	282	net.mobiletv.mobile.MainService	S282
137	com.cc.service.Hearttttt	S137	283	org.android.eldemo.dequevaPodcast.MyService	S283
138	com.cc.service.Int	S138	284	org.OpenUDID.OpenUDID_service	S284
139	com.cc.service.Ir	S139	285	org.par.ProximityAudio.LocationCheckService	S285
140	com.dreamstep.wMilitaryMeetDating.Server.C2DMClientReceiver	S140	286	org.simplelocker.MainService	S286
141	com.geinimi.c.c	S141	287	org.torproject.android.service.TorService	S287
142	com.google.android.smart.632799EAE241DDF1A8EA0DBB8C16E38Bcom_google_android_smart_McbainServicce	S142	288	personal.jhjeong.app.keepwifilite.UpdateService	S288
143	com.google.android.smart.McainService	S143	289	ru.droidlab.bogrpro.service.AutoupdateService	S289
144	com.iadpush.adp.BS	S144	290	ru.droidlab.bogrpro.service.QuotesUpdateService	S290
145	com.iadpush.adp.NS	S145	291	ws.coverme.im.model.push.GCMIntentService	S291
146	com.ivona.tts.voicelib.VoiceDownloaderService	S146	292	ws.coverme.im.service.CMCoreService	S292

API Calls Indexes

S #	API Calls	Id
1	connect	AP1
2	getContent	AP2
3	getWifiState	AP3
4	getNetworkInfo	AP4
5	getActiveNetworkInfo	AP5
6	LocationListener	AP6
7	requestLocationUpdates	AP7
8	getLastKnownLocation	AP8
9	getLine1Number	AP9
10	getDeviceId	AP10
11	openFileDescriptor	AP11
12	getInputStream	AP12
13	getSimSerialNumber	AP13
14	getSubscriberId	AP14
15	sendTextMessage	AP15

APPENDIX C

Used Pattern ID	Used features Pattern	Support Values	
		Botware	Benign
UP1	P1,P7,P17,B4,AP6,P19	0.97	0.03
UP2	P1,P5,P19,B4,AP19,S15,AP6	0.94	0.11
UP3	P1,P15,B11,AP17,P18,S10	0.92	0.08
UP4	P2,P5,P10,B6,AP6,S15,AP6,P5,A18	0.90	0.14
UP5	P2,P14,B18,AP16,S7	0.90	0.18
UP6	P1,P12,B20,AP9,S5,AP17	0.89	0.00
UP7	P1,P16,B17,AP7,S11,AP1,P2,S11	0.88	0.11
UP8	P1,P2,B19,AP15,S12,AP5,P16,S19,S7	0.79	0.07
UP9	P1,P20,B2,AP18,S8,AP8,P18	0.79	0.14
UP10	P1,P14,B10,AP12,S14,AP2,P7,S5	0.78	0.02
UP11	P1,P20,B16,AP14,S4,AP13,P14,A15	0.78	0.18
UP12	P1,P6,B16,AP9,S4,AP13,P6,S9,S2,A15	0.78	0.21
UP13	P2,P10,B3,AP5,S20,AP19,P3	0.77	0.18
UP14	P1,P8,S1,AP10,S12,AP5,P12	0.77	0.02
UP15	P1,P5,B5,AP15,S19,AP14,P16,S16	0.73	0.00
UP16	P3,P7,S1,AP14,S5,AP17	0.72	0.00
UP17	P1,P4,B8,AP1,S6,AP18,P10,S17,S11	0.72	0.00
UP18	P1,P13,B9,AP11,S11,AP1	0.72	0.02
UP19	P1,P6,B7,AP20	0.72	0.19
UP20	P1,P9,B13,AP4,S20,AP19,P11,S13	0.72	0.07
UP21	P1,P5,B17,AP20,S7,AP15,P15,S3	0.71	0.00
UP22	P1,P18,B14,AP1,S10,AP3,P10	0.70	0.06
UP23	P1,P13,B3,AP16,S18,AP12,P13,S9,S13	0.70	0.00
UP24	P1,P17,B13,AP17,S10,AP3,P1,S18	0.68	0.08
UP25	P1,P20,B20,AP11,B1,AP4,P9,S12,S17	0.68	0.00
UP26	P1,P4,B12,AP6,S3	0.67	0.19
UP27	P1,P11,B5,AP2,S8	0.65	0.00
UP28	P1,P2,B7,AP8,S16,AP10,P4	0.63	0.01
UP29	P1,P20,B12,AP3	0.63	0.00
UP30	P1,P3,B19,AP4,S2,AP9	0.62	0.06
UP31	P1,P8,B15,AP2,S13,AP16,P17	0.62	0.02
UP32	P1,P7,B6,AP19,S3,AP7,P19	0.62	0.00
UP33	P1,P16,B11,AP3,S17,AP11,P8,S3	0.61	0.00
UP34	P1,P12,B4,AP12,S9,AP20,P7,S5,S18	0.60	0.00
UP35	P1,P15,B2,AP8,S17,AP11	0.59	0.17
UP36	P1,P10,B10,AP18,S13,AP16	0.58	0.03
UP37	P1,P9,B18,AP13,S9,AP20,P20	0.56	0.00
UP38	P2,P3,B9,AP10,S19,AP14	0.54	0.14
UP39	P2,P18,B8,AP5,S2,AP9,P3,S2	0.53	0.17
UP40	P2,P17,B15,AP7,B1,AP4,P2	0.51	0.00
UP41	P1,P4,B8,AP1,S6,AP18,P10,S17,S14	0.50	0.18
UP42	P1,P7,S1,AP14,S5,AP19	0.50	0.03
UP43	P1,P12,B20,AP9,S5,AP20	0.50	0.20
UP44	P1,P4,B12,AP6,S5	0.50	0.01
UP45	P1,P17,B15,AP7,B1,AP4,P5	0.49	0.05
UP46	P1,P20,B2,AP18,S8,AP8,P21	0.49	0.07
UP47	P1,P12,B4,AP12,S9,AP20,P7,S5,S22	0.49	0.10

Used Pattern ID	Used features Pattern	Support Values	
		Botware	Benign
UP48	P1,P15,B2,AP8,S17,AP13	0.49	0.01
UP49	P1,P4,B12,AP6,S6	0.49	0.07
UP50	P1,P16,B11,AP3,S17,AP11,P8,S7	0.49	0.08
UP51	P1,P16,B11,AP3,S17,AP11,P8,S6	0.49	0.16
UP52	P1,P18,B14,AP1,S10,AP3,P12	0.48	0.13
UP53	P1,P6,B7,AP22	0.48	0.17
UP54	P1,P8,B15,AP2,S13,AP16,P18	0.48	0.20
UP55	P1,P7,B6,AP19,S3,AP7,P21	0.48	0.20
UP56	P1,P12,B4,AP12,S9,AP20,P7,S5,S18	0.48	0.03
UP57	P1,P5,B5,AP15,S19,AP14,P16,S19	0.47	0.05
UP58	P1,P20,B16,AP14,S4,AP13,P14,A17	0.47	0.11
UP59	P1,P3,B19,AP4,S2,AP9	0.47	0.08
UP60	P1,P13,B9,AP11,S11,AP3	0.47	0.09
UP61	P1,P11,B5,AP2,S10	0.46	0.10
UP62	P1,P2,B7,AP8,S16,AP10,P6	0.46	0.13
UP63	P1,P4,B8,AP1,S6,AP18,P10,S17,S12	0.46	0.19
UP64	P1,P18,B14,AP1,S10,AP3,P14	0.46	0.01
UP65	P1,P9,B13,AP4,S20,AP19,P11,S14	0.46	0.00
UP66	P1,P2,B7,AP8,S16,AP10,P8	0.46	0.05
UP67	P1,P20,B2,AP18,S8,AP8,P20	0.46	0.10
UP68	P1,P9,B18,AP13,S9,AP20,P21	0.46	0.10
UP69	P1,P14,B18,AP16,S11	0.46	0.05
UP70	P1,P2,B19,AP15,S12,AP5,P16,S19,S9	0.46	0.13
UP71	P1,P12,B20,AP9,S5,AP18	0.45	0.16
UP72	P1,P6,B16,AP9,S4,AP13,P6,S9,S2,A17	0.45	0.04
UP73	P1,P12,B20,AP9,S5,AP21	0.45	0.13
UP74	P1,P11,B5,AP2,S9	0.45	0.14
UP75	P1,P7,P17,B4,AP6,P21	0.45	0.17
UP76	P1,P17,B13,AP17,S10,AP3,P1,S20	0.45	0.08
UP77	P1,P17,B13,AP17,S10,AP3,P1,S18	0.45	0.07
UP78	P1,P2,B19,AP15,S12,AP5,P16,S19,S8	0.45	0.07
UP79	P1,P15,B2,AP8,S17,AP11	0.45	0.16
UP80	P2,P2,B19,A P25,S12,AP5, P26,S19,S9	0.44	0.02
UP81	P1,P16,B11,AP3,S17,AP11,P8,S5	0.44	0.20
UP82	P1,P9,B18,AP13,S9,AP20,P23	0.44	0.13
UP83	P1,P2,B19,AP15,S12,AP5,P16,S19,S10	0.44	0.01
UP84	P1,P16,B11,AP3,S17,AP11,P8,S3	0.44	0.10
UP85	P1,P14,B18,AP16,S9	0.44	0.04
UP86	P1,P13,B3,AP16,S18,AP12,P13,S9,S16	0.44	0.20
UP87	P1,P5,B17,AP20,S7,AP15,P15,S6	0.44	0.07
UP88	P1,P5,B5,AP15,S19,AP14,P16,S20	0.44	0.07
UP89	P1,P13,B9,AP11,S11,AP5	0.44	0.15
UP90	P1,P7,P17,B4,AP6,P21	0.44	0.05
UP91	P1,P12,B20,AP9,S5,AP17	0.44	0.05
UP92	P1,P11,B5,AP2,S11	0.44	0.19
UP93	P1,P8,S1,AP10,S12,AP5,P16	0.43	0.17
UP94	P1,P12,B4,AP12,S9,AP20,P7,S5,S21	0.43	0.14
UP95	P1,P17,B13,AP17,S10,AP3,P1,S21	0.43	0.14
UP96	P1,P7,P17,B4,AP6,P19	0.43	0.11
UP97	P1,P3,B19,AP4,S2,AP11	0.43	0.09

Used Pattern ID	Used features Pattern	Support Values	
		Botware	Benign
UP98	P1,P10,B3,AP5,S20,AP19,P6	0.43	0.07
UP99	P1,P5,P19,B4,AP19,S15,AP7	0.43	0.12
UP100	P1,P9,B13,AP4,S20,AP19,P11,S16	0.43	0.16
UP101	P1,P16,B17,AP7,S11,AP1,P2,S11	0.42	0.04
UP102	P1,P5,P10,B6,AP6,S15,AP6,P5,A20	0.42	0.18
UP103	P1,P11,B5,AP2,S10	0.42	0.14
UP104	P1,P20,B16,AP14,S4,AP13,P14,A19	0.42	0.04
UP105	P1,P14,B10,AP12,S14,AP2,P7,S6	0.42	0.17
UP106	P1,P9,B13,AP4,S20,AP19,P11,S14	0.41	0.05
UP107	P1,P5,P19,B4,AP19,S15,AP10	0.41	0.15
UP108	P1,P20,B12,AP5	0.41	0.17
UP109	P1,P5,P19,B4,AP19,S15,AP8	0.40	0.02
UP110	P1,P5,B17,AP20,S7,AP15,P15,S4	0.40	0.15
UP111	P1,P4,B12,AP6,S6	0.40	0.03
UP112	P1,P17,B15,AP7,B1,AP4,P2	0.40	0.13
UP113	P1,P8,B15,AP2,S13,AP16,P21	0.40	0.18
UP114	P1,P5,P10,B6,AP6,S15,AP6,P5,A19	0.39	0.02
UP115	P1,P9,B18,AP13,S9,AP20,P22	0.39	0.05
UP116	P1,P17,B13,AP17,S10,AP3,P1,S20	0.39	0.10
UP117	P1,P20,B16,AP14,S4,AP13,P14,A16	0.39	0.12
UP118	P1,P14,B10,AP12,S14,AP2,P7,S8	0.39	0.07
UP119	P1,P16,B17,AP7,S11,AP1,P2,S14	0.39	0.14
UP120	P1,P17,B13,AP17,S10,AP3,P1,S22	0.39	0.10
UP121	P1,P20,B20,AP11,B1,AP4,P9,S12,S20	0.39	0.10
UP122	P1,P10,B10,AP18,S13,AP17	0.39	0.16
UP123	P1,P14,B18,AP16,S10	0.38	0.14
UP124	P1,P10,B3,AP5,S20,AP19,P5	0.38	0.01
UP125	P1,P8,B15,AP2,S13,AP16,P17	0.38	0.16
UP126	P1,P6,B16,AP9,S4,AP13,P6,S9,S2,A19	0.38	0.04
UP127	P1,P12,B20,AP9,S5,AP20	0.38	0.16
UP128	P1,P9,B13,AP4,S20,AP19,P11,S17	0.38	0.13
UP129	P1,P15,B2,AP8,S17,AP14	0.38	0.16
UP130	P1,P8,B15,AP2,S13,AP16,P19	0.38	0.19
UP131	P1,P20,B2,AP18,S8,AP8,P20	0.38	0.00
UP132	P1,P4,B12,AP6,S5	0.38	0.13
UP133	P1,P16,B17,AP7,S11,AP1,P2,S14	0.37	0.10
UP134	P1,P13,B9,AP11,S11,AP1	0.37	0.15
UP135	P1,P2,B19,AP15,S12,AP5,P16,S19,S7	0.37	0.11
UP136	P1,P18,B8,AP5,S2,AP9,P3,S4	0.37	0.04
UP137	P1,P14,B10,AP12,S14,AP2,P7,S5	0.37	0.13
UP138	P1,P18,B14,AP1,S10,AP3,P11	0.37	0.14
UP139	P1,P5,P19,B4,AP19,S15,AP9	0.37	0.20
UP140	P1,P8,B15,AP2,S13,AP16,P19	0.36	0.05
UP141	P2, P25,B11,A P27, P28,S13	0.36	0.17
UP142	P2,P7,P17,B4,AP6,P22	0.36	0.17
UP143	P1,P5,B5,AP15,S19,AP14,P16,S17	0.36	0.20
UP144	P1,P7,S1,AP14,S5,AP17	0.36	0.11
UP145	P1,P12,B4,AP12,S9,AP20,P7,S5,S20	0.36	0.06
UP146	P1,P13,B9,AP11,S11,AP3	0.36	0.18
UP147	P1,P16,B11,AP3,S17,AP11,P8,S4	0.36	0.12

Used Pattern ID	Used features Pattern	Support Values	
		Botware	Benign
UP148	P1,P14,B10,AP12,S14,AP2,P7,S6	0.35	0.16
UP149	P1,P5,P19,B4,AP19,S15,AP6	0.35	0.16
UP150	P1,P3,B9,AP10,S19,AP16	0.35	0.20
UP151	P1,P20,B20,AP11,B1,AP4,P9,S12,S19	0.35	0.01
UP152	P1,P20,B20,AP11,B1,AP4,P9,S12,S17	0.35	0.10
UP153	P1,P3,B9,AP10,S19,AP15	0.35	0.00
UP154	P1,P16,B17,AP7,S11,AP1,P2,S15	0.35	0.18
UP155	P1,P14,B18,AP16,S8	0.35	0.02
UP156	P1,P13,B3,AP16,S18,AP12,P13,S9,S17	0.35	0.16
UP157	P1,P5,B5,AP15,S19,AP14,P16,S19	0.34	0.12
UP158	P1,P3,B19,AP4,S2,AP10	0.34	0.05
UP159	P1,P6,B7,AP20	0.34	0.00
UP160	P1,P7,S1,AP14,S5,AP20	0.34	0.00
UP161	P1,P12,B20,AP9,S5,AP19	0.34	0.12
UP162	P1,P20,B2,AP18,S8,AP8,P19	0.34	0.14
UP163	P1,P5,B5,AP15,S19,AP14,P16,S18	0.34	0.17
UP164	P1,P17,B15,AP7,B1,AP4,P4	0.34	0.02
UP165	P1,P16,B17,AP7,S11,AP1,P2,S13	0.34	0.04
UP166	P2,P4,B8,AP1,S6,AP18,P10,S17,S13	0.33	0.11
UP167	P1,P7,S1,AP14,S5,AP19	0.33	0.12
UP168	P1,P15,B11,AP17,P18,S12	0.33	0.02
UP169	P1,P4,B12,AP6,S7	0.33	0.09
UP170	P1,P10,B10,AP18,S13,AP16	0.33	0.11
UP171	P1,P7,S1,AP14,S5,AP19	0.33	0.10
UP172	P1,P8,B15,AP2,S13,AP16,P20	0.33	0.07
UP173	P1,P20,B2,AP18,S8,AP8,P22	0.32	0.12
UP174	P2,P10,B10,AP18,S13,AP19	0.32	0.14
UP175	P1,P6,B16,AP9,S4,AP13,P6,S9,S2,A17	0.32	0.18
UP176	P1,P5,B17,AP20,S7,AP15,P15,S5	0.32	0.19
UP177	P1,P5,B17,AP20,S7,AP15,P15,S4	0.32	0.08
UP178	P1,P3,B9,AP10,S19,AP15	0.32	0.00
UP179	P1,P14,B10,AP12,S14,AP2,P7,S7	0.32	0.19
UP180	P1,P20,B12,AP5	0.32	0.03
UP181	P1,P14,B18,AP16,S9	0.32	0.01
UP182	P1,P7,P17,B4,AP6,P20	0.32	0.07
UP183	P1,P15,B11,AP17,P18,S12	0.32	0.11
UP184	P1,P17,B13,AP17,S10,AP3,P1,S21	0.32	0.12
UP185	P1,P13,B9,AP11,S11,AP4	0.31	0.07
UP186	P1,P10,B3,AP5,S20,AP19,P6	0.31	0.20
UP187	P1,P8,S1,AP10,S12,AP5,P13	0.31	0.15
UP188	P1,P5,P19,B4,AP19,S15,AP8	0.31	0.20
UP189	P1,P16,B11,AP3,S17,AP11,P8,S5	0.31	0.13
UP190	P1,P8,S1,AP10,S12,AP5,P12	0.31	0.12
UP191	P1,P6,B7,AP21	0.31	0.07
UP192	P1,P12,B4,AP12,S9,AP20,P7,S5,S19	0.31	0.07
UP193	P1,P5,P10,B6,AP6,S15,AP6,P5,A20	0.31	0.11
UP194	P1,P2,B7,AP8,S16,AP10,P6	0.31	0.04
UP195	P1,P10,B10,AP18,S13,AP18	0.31	0.06
UP196	P1,P18,B14,AP1,S10,AP3,P10	0.31	0.20
UP197	P1,P13,B3,AP16,S18,AP12,P13,S9,S13	0.31	0.06

Used Pattern ID	Used features Pattern	Support Values	
		Botware	Benign
UP198	P1,P8,S1,AP10,S12,AP5,P14	0.30	0.04
UP199	P1,P12,B4,AP12,S9,AP20,P7,S5,S21	0.30	0.16
UP200	P1,P15,B11,AP17,P18,S13	0.29	0.03
UP201	P1,P20,B12,AP7	0.29	0.20
UP202	P1,P6,B7,AP24	0.29	0.04
UP203	P1,P7,B6,AP19,S3,AP7,P22	0.29	0.18
UP204	P1,P16,B11,AP3,S17,AP11,P8,S6	0.29	0.06
UP205	P1,P4,B8,AP1,S6,AP18,P10,S17,S14	0.28	0.03
UP206	P1,P20,B12,AP6	0.28	0.13
UP207	P1,P9,B13,AP4,S20,AP19,P11,S13	0.28	0.16
UP208	P1,P10,B3,AP5,S20,AP19,P5	0.27	0.12
UP209	P1,P20,B20,AP11,B1,AP4,P9,S12,S18	0.27	0.09
UP210	P1,P3,B9,AP10,S19,AP17	0.26	0.08
UP211	P1,P5,P19,B4,AP19,S15,AP8	0.26	0.14
UP212	P1,P12,B4,AP12,S9,AP20,P7,S5,S20	0.26	0.17
UP213	P1,P15,B11,AP17,P18,S11	0.26	0.17
UP214	P1,P18,B14,AP1,S10,AP3,P12	0.26	0.13
UP215	P1,P8,S1,AP10,S12,AP5,P13	0.26	0.13
UP216	P1,P3,B19,AP4,S2,AP12	0.26	0.17
UP217	P1,P16,B17,AP7,S11,AP1,P2,S13	0.26	0.07
UP218	P1,P9,B13,AP4,S20,AP19,P11,S15	0.25	0.12
UP219	P1,P20,B20,AP11,B1,AP4,P9,S12,S20	0.25	0.17
UP220	P1,P3,B19,AP4,S2,AP11	0.25	0.07
UP221	P1,P9,B18,AP13,S9,AP20,P21	0.24	0.03
UP222	P1,P10,B3,AP5,S20,AP19,P4	0.24	0.15
UP223	P1,P13,B9,AP11,S11,AP2	0.24	0.20
UP224	P1,P11,B5,AP2,S10	0.24	0.11
UP225	P1,P20,B16,AP14,S4,AP13,P14,A18	0.24	0.13
UP226	P1,P10,B10,AP18,S13,AP17	0.24	0.16
UP227	P1,P5,B17,AP20,S7,AP15,P15,S7	0.24	0.06
UP228	P1,P18,B14,AP1,S10,AP3,P13	0.23	0.17
UP229	P1,P7,P17,B4,AP6,P23	0.23	0.06
UP230	P1,P17,B15,AP7,B1,AP4,P3	0.23	0.10
UP231	P1,P12,B4,AP12,S9,AP20,P7,S5,S19	0.23	0.14
UP232	P1,P14,B18,AP16,S8	0.23	0.06
UP233	P1,P18,B14,AP1,S10,AP3,P12	0.22	0.11
UP234	P1,P16,B17,AP7,S11,AP1,P2,S12	0.22	0.16
UP235	P1,P16,B17,AP7,S11,AP1,P2,S12	0.22	0.01
UP236	P1,P4,B12,AP6,S4	0.22	0.19
UP237	P1,P5,B5,AP15,S19,AP14,P16,S17	0.22	0.09
UP238	P1,P20,B20,AP11,B1,AP4,P9,S12,S21	0.22	0.17
UP239	P1,P5,B17,AP20,S7,AP15,P15,S6	0.22	0.09
UP240	P1,P14,B10,AP12,S14,AP2,P7,S7	0.22	0.11
UP241	P1,P9,B18,AP13,S9,AP20,P20	0.22	0.07
UP242	P1,P5,P19,B4,AP19,S15,AP7	0.21	0.19
UP243	P1,P7,P17,B4,AP6,P22	0.21	0.15
UP244	P1,P14,B10,AP12,S14,AP2,P7,S7	0.21	0.07
UP245	P1,P20,B20,AP11,B1,AP4,P9,S12,S18	0.21	0.08
UP246	P1,P17,B13,AP17,S10,AP3,P1,S20	0.21	0.13
UP247	P1,P8,S1,AP10,S12,AP5,P14	0.20	0.15

Used Pattern ID	Used features Pattern	Support Values	
		Botware	Benign
UP248	P1,P7,B6,AP19,S3,AP7,P20	0.20	0.02
UP249	P1,P4,B8,AP1,S6,AP18,P10,S17,S12	0.20	0.05
UP250	P1,P4,B8,AP1,S6,AP18,P10,S17,S11	0.20	0.06
UP251	P1,P8,S1,AP10,S12,AP5,P14	0.20	0.00
UP252	P1,P5,P10,B6,AP6,S15,AP6,P5,A22	0.20	0.00
UP253	P1,P5,B5,AP15,S19,AP14,P16,S16	0.20	0.07
UP254	P1,P6,B16,AP9,S4,AP13,P6,S9,S2,A18	0.19	0.10
UP255	P1,P11,B5,AP2,S11	0.19	0.00
UP256	P1,P9,B13,AP4,S20,AP19,P11,S15	0.19	0.04
UP257	P1,P7,B6,AP19,S3,AP7,P22	0.19	0.19
UP258	P1,P20,B16,AP14,S4,AP13,P14,A17	0.18	0.07
UP259	P1,P11,B5,AP2,S8	0.18	0.12
UP260	P1,P2,B7,AP8,S16,AP10,P5	0.18	0.15
UP261	P1,P5,B17,AP20,S7,AP15,P15,S3	0.18	0.17
UP262	P1,P6,B7,AP23	0.18	0.03
UP263	P1,P8,S1,AP10,S12,AP5,P15	0.18	0.03
UP264	P1,P16,B17,AP7,S11,AP1,P2,S13	0.18	0.02
UP265	P1,P7,B6,AP19,S3,AP7,P21	0.18	0.09
UP266	P1,P20,B2,AP18,S8,AP8,P19	0.17	0.14
UP267	P1,P12,B20,AP9,S5,AP18	0.17	0.17
UP268	P1,P10,B3,AP5,S20,AP19,P5	0.17	0.00
UP269	P1,P3,B9,AP10,S19,AP14	0.17	0.06
UP270	P1,P15,B2,AP8,S17,AP12	0.17	0.01
UP271	P1,P14,B10,AP12,S14,AP2,P7,S8	0.17	0.19
UP272	P1,P12,B4,AP12,S9,AP20,P7,S5,S20	0.17	0.19
UP273	P1,P2,B7,AP8,S16,AP10,P5	0.17	0.14
UP274	P1,P4,B12,AP6,S5	0.17	0.19
UP275	P1,P20,B12,AP5	0.17	0.11
UP276	P1,P4,B12,AP6,S3	0.16	0.17
UP277	P1,P15,B11,AP17,P18,S10	0.16	0.10
UP278	P1,P2,B7,AP8,S16,AP10,P7	0.16	0.02
UP279	P1,P18,B8,AP5,S2,AP9,P3,S3	0.16	0.08
UP280	P1,P6,B7,AP22	0.15	0.01
UP281	P1,P7,B6,AP19,S3,AP7,P23	0.15	0.18
UP282	P1,P6,B16,AP9,S4,AP13,P6,S9,S2,A16	0.15	0.20
UP283	P1,P18,B8,AP5,S2,AP9,P3,S3	0.15	0.17
UP284	P1,P13,B3,AP16,S18,AP12,P13,S9,S14	0.15	0.07
UP285	P1,P2,B7,AP8,S16,AP10,P7	0.15	0.16
UP286	P1,P6,B7,AP23	0.15	0.19
UP287	P1,P9,B13,AP4,S20,AP19,P11,S15	0.15	0.12
UP288	P1,P7,S1,AP14,S5,AP21	0.15	0.05
UP289	P1,P10,B10,AP18,S13,AP18	0.14	0.10
UP290	P1,P7,B6,AP19,S3,AP7,P21	0.14	0.11
UP291	P1,P8,B15,AP2,S13,AP16,P18	0.14	0.07
UP292	P1,P20,B20,AP11,B1,AP4,P9,S12,S19	0.14	0.07
UP293	P1,P6,B16,AP9,S4,AP13,P6,S9,S2,A17	0.14	0.10
UP294	P1,P13,B9,AP11,S11,AP4	0.14	0.02
UP295	P1,P7,B6,AP19,S3,AP7,P20	0.13	0.01
UP296	P1,P5,P10,B6,AP6,S15,AP6,P5,A20	0.13	0.06
UP297	P1,P2,B19,AP15,S12,AP5,P16,S19,S8	0.13	0.18

Used Pattern ID	Used features Pattern	Support Values	
		Botware	Benign
UP298	P1,P13,B3,AP16,S18,AP12,P13,S9,S15	0.13	0.15
UP299	P1,P17,B13,AP17,S10,AP3,P1,S19	0.12	0.08
UP300	P1,P20,B16,AP14,S4,AP13,P14,A17	0.12	0.12
UP301	P1,P5,P10,B6,AP6,S15,AP6,P5,A19	0.12	0.16
UP302	P1,P7,B6,AP19,S3,AP7,P19	0.12	0.03
UP303	P1,P5,P10,B6,AP6,S15,AP6,P5,A21	0.12	0.07
UP304	P1,P10,B3,AP5,S20,AP19,P4	0.12	0.18
UP305	P1,P15,B11,AP17,P18,S11	0.12	0.04
UP306	P1,P13,B3,AP16,S18,AP12,P13,S9,S15	0.11	0.00
UP307	P1,P14,B18,AP16,S10	0.11	0.09
UP308	P1,P20,B16,AP14,S4,AP13,P14,A16	0.11	0.02
UP309	P1,P6,B16,AP9,S4,AP13,P6,S9,S2,A18	0.11	0.19
UP310	P1,P18,B14,AP1,S10,AP3,P13	0.11	0.06
UP311	P1,P2,B19,AP15,S12,AP5,P16,S19,S11	0.10	0.07
UP312	P1,P6,B16,AP9,S4,AP13,P6,S9,S2,A15	0.10	0.09
UP313	P1,P5,B17,AP20,S7,AP15,P15,S5	0.10	0.09
UP314	P1,P2,B7,AP8,S16,AP10,P6	0.10	0.10
UP315	P1,P16,B11,AP3,S17,AP11,P8,S5	0.10	0.16
UP316	P1,P2,B7,AP8,S16,AP10,P4	0.10	0.14
UP317	P1,P3,B19,AP4,S2,AP13	0.09	0.20
UP318	P1,P20,B2,AP18,S8,AP8,P21	0.09	0.08
UP319	P1,P3,B19,AP4,S2,AP10	0.09	0.18
UP320	P1,P4,B8,AP1,S6,AP18,P10,S17,S13	0.09	0.11
UP321	P1,P18,B8,AP5,S2,AP9,P3,S2	0.09	0.04
UP322	P2,P16,B11,AP3,S17,AP11,P8,S4	0.08	0.01
UP323	P1,P14,B10,AP12,S14,AP2,P7,S9	0.08	0.05
UP324	P1,P3,B9,AP10,S19,AP16	0.08	0.02
UP325	P1,P18,B8,AP5,S2,AP9,P3,S4	0.08	0.07
UP326	P1,P5,P19,B4,AP19,S15,AP9	0.08	0.10
UP327	P1,P7,P17,B4,AP6,P20	0.07	0.05
UP328	P1,P11,B5,AP2,S12	0.07	0.03
UP329	P1,P12,B20,AP9,S5,AP19	0.07	0.14
UP330	P1,P8,B15,AP2,S13,AP16,P20	0.07	0.03
UP331	P1,P15,B2,AP8,S17,AP13	0.07	0.10
UP332	P1,P17,B13,AP17,S10,AP3,P1,S19	0.07	0.04
UP333	P1,P14,B18,AP16,S9	0.06	0.03
UP334	P1,P9,B13,AP4,S20,AP19,P11,S16	0.06	0.07
UP335	P1,P17,B15,AP7,B1,AP4,P4	0.06	0.05
UP336	P1,P4,B8,AP1,S6,AP18,P10,S17,S13	0.06	0.17
UP337	P1,P9,B18,AP13,S9,AP20,P22	0.05	0.20
UP338	P1,P15,B11,AP17,P18,S12	0.05	0.15
UP339	P1,P20,B12,AP4	0.05	0.12
UP340	P1,P13,B9,AP11,S11,AP2	0.04	0.15
UP341	P1,P5,B17,AP20,S7,AP15,P15,S5	0.04	0.05
UP342	P1,P3,B19,AP4,S2,AP11	0.04	0.05
UP343	P1,P15,B2,AP8,S17,AP12	0.04	0.04
UP344	P1,P3,B19,AP4,S2,AP12	0.04	0.13
UP345	P1,P20,B20,AP11,B1,AP4,P9,S12,S19	0.04	0.05
UP346	P2,P2,B19,AP15,S12,AP5,P16,S19,S9	0.03	0.13
UP347	P1,P5,P10,B6,AP6,S15,AP6,P5,A21	0.03	0.02

Used Pattern ID	Used features Pattern	Support Values	
		Botware	Benign
UP348	P1,P5,B5,AP15,S19,AP14,P16,S18	0.03	0.20
UP349	P1,P20,B12,AP6	0.03	0.01
UP350	P1,P20,B12,AP3	0.03	0.14
UP351	P1,P10,B3,AP5,S20,AP19,P7	0.03	0.16
UP352	P1,P4,B12,AP6,S4	0.03	0.19
UP353	P1,P8,B15,AP2,S13,AP16,P19	0.03	0.00
UP354	P1,P14,B18,AP16,S7	0.03	0.07
UP355	P1,P5,B5,AP15,S19,AP14,P16,S18	0.02	0.07
UP356	P1,P17,B15,AP7,B1,AP4,P3	0.02	0.03
UP357	P1,P6,B16,AP9,S4,AP13,P6,S9,S2,A16	0.02	0.13
UP358	P1,P7,S1,AP14,S5,AP20	0.02	0.13
UP359	P1,P20,B12,AP4	0.02	0.02
UP360	P1,P20,B16,AP14,S4,AP13,P14,A18	0.02	0.13
UP361	P1,P18,B8,AP5,S2,AP9,P3,S5	0.02	0.13
UP362	P1,P2,B19,AP15,S12,AP5,P16,S19,S10	0.02	0.00
UP363	P1,P18,B14,AP1,S10,AP3,P11	0.02	0.10
UP364	P1,P13,B3,AP16,S18,AP12,P13,S9,S16	0.02	0.03
UP365	P2,P20,B2,A P28,S8,AP8,P20	0.01	0.16
UP366	P1,P11,B5,AP2,S9	0.01	0.10
UP367	P1,P4,B8,AP1,S6,AP18,P10,S17,S15	0.01	0.11
UP368	P1,P7,S1,AP14,S5,AP18	0.01	0.10
UP369	P1,P6,B7,AP22	0.01	0.04
UP370	P1,P20,B2,AP18,S8,AP8,P18	0.01	0.07
UP371	P1,P7,S1,AP14,S5,AP18	0.01	0.10
UP372	P1,P13,B9,AP11,S11,AP3	0.00	0.17
UP373	P1,P15,B11,AP17,P18,S14	0.00	0.13
UP374	P1,P13,B3,AP16,S18,AP12,P13,S9,S14	0.00	0.00
UP375	P1,P5,P10,B6,AP6,S15,AP6,P5,A18	0.00	0.06
UP376	P1,P10,B3,AP5,S20,AP19,P3	0.00	0.05
UP377	P1,P20,B16,AP14,S4,AP13,P14,A15	0.00	0.10
UP378	P1,P7,P17,B4,AP6,P21	0.00	0.11
UP379	P1,P6,B7,AP21	0.00	0.09
UP380	P1,P13,B3,AP16,S18,AP12,P13,S9,S15	0.00	0.17
UP381	P1,P8,S1,AP10,S12,AP5,P15	0.00	0.02
UP382	P1,P12,B20,AP9,S5,AP19	0.00	0.01

APPENDIX D

Overall Experimental Classification Results for Drebin Dataset

Scheme	Folds	Training	# of Correct in % age	# of Incorrect in % age	# of Correct	# of Incorrect	Total True Positive	Total False Positive	Total True Negative	Total False Negative
Support Vector Machine	2	280	66.00	34.00	183	97	70	25	75	13
	3	280	69.00	31.10	192	88	72	22	80	18
	4	280	77.20	22.80	215	65	112	14	70	19
	5	280	83.80	16.30	234	46	98	13	82	41
	6	280	84.80	15.30	237	43	102	16	96	23
	7	280	84.80	15.30	237	43	109	14	112	2
	8	280	87.10	12.90	243	37	94	15	102	32
	9	280	86.00	14.10	240	40	106	19	95	20
	10	280	87.40	12.60	244	36	100	25	92	27
J48	2	280	90.50	9.50	253	27	94	26	105	28
	3	280	91.80	8.30	257	24	98	25	111	23
	4	280	91.80	8.30	257	24	108	22	120	7
	5	280	92.80	7.30	260	21	117	21	92	30
	6	280	91.50	8.50	256	24	112	21	93	30
	7	280	90.80	9.20	254	26	109	28	103	14
	8	280	89.60	10.40	250	30	105	21	118	6
	9	280	90.30	9.70	252	28	109	21	110	12
	10	280	91.30	8.70	255	25	117	26	94	18

Scheme	**Folds**	**Training**	**# of Correct in % age**	**# of Incorrect in % age**	**# of Correct**	**# of Incorrect**	**Total True Positive**	**Total False Positive**	**Total True Negative**	**Total False Negative**
Random Forest	**2**	**280**	**96.70**	**7.30**	**259**	**21**	**110**	**29**	**102**	**18**
	3	280	98.40	6.60	261	19	109	22	112	18
	4	280	98.90	7.00	260	20	114	18	109	19
	5	280	98.90	6.10	262	18	119	13	109	21
	6	280	97.70	6.30	262	18	109	15	115	23
	7	280	98.70	6.30	262	18	157	9	92	4
	8	280	98.80	7.30	259	21	112	13	102	32
	9	280	98.70	6.30	262	18	116	17	109	20
	10	280	97.20	5.80	263	17	119	23	109	12
Simple Logistic Regression	2	280	87.00	13.10	243	37	115	12	102	14
	3	280	87.70	12.40	245	35	104	15	111	15
	4	280	86.40	13.60	241	39	109	12	108	12
	5	280	86.90	13.10	243	37	117	13	92	21
	6	280	87.40	12.60	244	36	112	15	95	22
	7	280	86.90	13.10	243	37	109	25	97	12
	8	280	87.10	12.90	243	37	105	18	116	4
	9	280	87.10	12.90	243	37	111	17	105	10
	10	280	86.90	13.10	243	37	115	22	94	12

Scheme	Fol ds	Traini ng	# of Correct in % age	# of Incorrect in % age	# of Correct	# of Incorrect	Total True Positive	Total False Positive	Total True Negative	Total False Negative
Naïve Bayes	2	280	86.90	13.10	243	37	104	15	107	17
	3	280	87.60	12.40	245	35	112	12	117	4
	4	280	86.40	13.60	241	39	110	17	102	12
	5	280	86.90	13.10	243	37	92	21	103	27
	6	280	87.40	12.60	244	36	109	28	100	7
	7	280	86.90	13.10	243	37	105	14	118	6
	8	280	87.10	12.90	243	37	109	11	111	12
	9	280	87.10	12.90	243	37	109	22	100	12
	10	280	87.10	12.90	243	37	105	22	104	12